GMIS: AN EXPERIMENTAL SYSTEM FOR DATA MANAGEMENT AND ANALYSIS

by

John J. Donovan and Henry D. Jacoby
M.I.T.

Center for Information Systems Research
in Association with M.I.T. Energy Laboratory

September 1975

REPORT CISR-16
SLOAN WP 811-75

GMIS: AN EXPERIMENTAL SYSTEM FOR DATA MANAGEMENT AND ANALYSIS

by

John J. Donovan and Henry D. Jacoby
M.I.T.

Center for Information Systems Research
in Association with M.I.T. Energy Laboratory

September 1975

REPORT CISR-16
SLOAN WP 811-75

OUTLINE

ACKNOWLEDGEMENT

1. History and Purpose of GMIS

2. Overview of the System Architecture

    2.1 Structured English Query Language (SEQUEL)

    2.2 Multi-User Transaction Interface

    2.3 User Interfaces

3. Sample Application of GMIS

    3.1 Data Manipulation

        3.1.1 Data Definition Facility

        3.1.2 Bulk Loading Facility

        3.1.3 System Inquiry Facility

        3.1.4 Query Facility

    3.2 Modeling and Analytical Functions

        3.2.1 Validating Data

        3.2.2 Reporting

        3.2.3 Modeling

        3.2.4 Stand-Alone Modeling Facility

4. Details of the GMIS Design

    4.1 The Use of VM in the Software Architecture

        4.1.1 Communication between VM's

        4.1.2 Extensions of the Architecture

        4.1.3 Degradation of Variable Cost with Multiple VM Operation

4.2 Hierarchical Approach

4.3 Relational Technology

4.3.1 Advantages of the Relational Approach

4.3.2 Basics of Relational Operators

5. Further Research

5.1 Computer and Management Science Research

5.2 Studies of the Economics of Information System Design

ACKNOWLEDGEMENT

The research reported here is being carried out as a joint project of the M.I.T. Energy Laboratory and the Center for Information Systems Research of the Alfred P. Sloan School of Management at M.I.T. The work was made possible by support from an M.I.T./IBM Joint Study Agreement, the New England Regional Commission [Contract No. 10530680], the Federal Energy Administration [Contract No. 14-01-001-2040], and M.I.T. internal funds.

Members of the IBM Cambridge Scientific Center and of the IBM Research Laboratory of San Jose have greatly contributed to this work. Those at IBM whom we are particularly indebted to are: Dr. Ray Fessel for his ingenious programming guidance, to Dr. Stuart Greenberg for his help with VM, and to Dr. Frank King and his group for their work in implementing SEQUEL and for their cooperation and responsiveness in adapting this experimental system to meet the operational needs of GMIS.

At the New England Regional Commission thanks are due to Robert Keating for his instructive guidance in the application of GMIS to energy problems facing New England.

We wish to thank Dr. Robert Goldberg of Harvard for his comments in reviewing this document.

We also wish to recognize the assistance of the several M.I.T. students who have contributed to the research. In particular, credit is due to Louis Gutentag, who has borne the major responsibility for the implementation of GMIS, and to Marvin Essrig, Peter DiGiammarino, and John Maglio, who assisted in the preparation of this document.

GMIS: AN EXPERIMENTAL SYSTEM FOR DATA MANAGEMENT AND ANALYSIS

John J. Donovan
Henry D. Jacoby

How many people would climb aboard a trans-Atlantic flight if they thought the airline lacked the capability to process volumes of weather and traffic data, and to plan a safe route? Not many, for most of us have come to expect that the very best information processing services will be applied in this circumstance. Yet public policymakers and corporate executives are regularly faced with far more complex and serious problems (perhaps with risks that are less immediate and obvious), and must make decisions without the capacity to manage and analyze the pertinent information. This happens for several reasons: Circumstances arise unexpectedly, and under current technology there simply is not time to construct the necessary software, or decisions may not occur regularly enough to justify the cost of a normal information management system, particularly when its useful life may be cut short by changing circumstances. In this paper we report on an effort to design and implement tools appropriate to this circumstance.

The system under development is called GMIS (Generalized Management Information System), and we present the underlying architecture of the system and its rationale, along with a sample demonstration of its characteristics. We begin, in Section 1, with a brief history of the effort and a summary statement of what the system is designed to do. In order to give a quick summary of how the system works, Section 2 is an overview of the software architecture; and then Section 3 uses a sample application

to an energy analysis problem in order to describe how the system is used and what some of its more important features are. For the reader interested in details we return in Section 4 to more discussion of the techniques and methods used in building GMIS. Finally, since this is a report of research in process, a summary of topics of continuing research is given in Section 5.

1. HISTORY AND PURPOSE OF GMIS

GMIS is being developed at the M.I.T. Sloan School's Center for Information Systems Research and the M.I.T. Energy Laboratory in conjunction with IBM. The project started in 1973 based on ongoing research in the Sloan School on file systems [Madnick, 1970] and operating systems [Donovan, 1972; Madnick and Donovan, 1974]. However, it has been the urgency of particular applications to energy problems that has shaped the work and quickened its pace.

During the energy crisis of the winter of 1973/74, policymakers in New England were handicapped by a lack of information about the region's energy economy. In response to this circumstance, the New England Regional Commission (NERCOM) initiated a project to develop a New England Energy Management Information System (NEEMIS). The initial plan was to develop a "crisis management" system to assist in the handling of fuel oil allocation, but over time (though the original function remains an important one) the needs have grown and the emphasis shifted. Problems of the economic impact of high oil prices have taken on more importance along with policies and programs to foster energy conservation. New issues have arisen concerning the location of major energy facilities, bringing a need for analysis of associated economic and environmental issues.

Growing experience with the data also brought more demands on the system design. The data are of varying quality; data collection procedures are changing over time, with series being dropped and added and definitions being revised. The requirements for protection have proved complex, for they vary with levels of aggregation and time. (For example, an oil company

may be willing to give out data on its aggregate transactions, but not on details that may help a competitor.) Finally, the need for a facility to apply various analytical models to the data has become more apparent.

In this circumstance, our approach has been to develop a general set of tools for speedy construction and easy modification of management information systems. Essentially, the need is for a software facility suitable for situations where the problem addressed is constantly changing, or where an information system is in its formulative stages and users are unable to specify exactly what they want the system to do, or precisely what the data streams will look like in the future.

To meet these requirements, certain characteristics of the system seem essential: it needs to be multi-user and interactive; it should be capable of storing, validating, and retrieving data; and it ought to have the capability to respond to changing data and data structure, and to varying protection requirements. It should provide tools for constructing analytical and statistical models to be applied to the data, but a facility to construct these models from scratch appears insufficient. Many economists and modelers have strong preferences for particular modeling facilities such as TROLL [NBER, 1973], XSIM [Dynamics Association, 1974], TSP [Hall, 1975], PL/I, EPLAN [Schober, 1974], and FORTRAN; large investments have been made in packages using these languages, and access to these facilities can save tremendous costs in retraining personnel and converting existing models.

Existing commerical data base systems -- e.g., IMS [IBM, 1968], DBTG [Association for Computing Machinery, 1971], System 2000 [MRI Systems, 1974], TOTAL [Cincom Systems, Inc., 1974] etc. -- have proved their usefulness

in particular applications. But none has the range of desired charactersitics outlined above. Some are lacking the statistical and modeling packages, not all are interactive, and not all can allow multiple users to access the same data base. Most important, none was designed for a changing environment. As detailed below, the GMIS system has taken a long step in this direction. Using this facility, it is possible to construct an information system in a matter of days. For example, in the course of work on the NEEMIS System, chages in the New England energy situation made it necessary to reconstruct the entire data base five times in one month during the summer of 1975 -- once to incorporate additional data in existing data series, twice for efficiency reasons, and twice because new data and models had to be added as new problems became apparent.

In the sections that follow, we give a brief overview of the architecture of the GMIS system and then illustrate the system characteristics by means of an example drawn from one of its energy applications. For the reader interested in the details of software design, the discussion goes on to cover more of the details of the system and its various components. Since the discussion cannot cover all aspects of the system, however, it is useful to summarize the requirements that the GMIS system has been designed to meet. First, in the area of data management the current system has the following features:

- it allows on-line interactive data management as well as a batch facility;
- it allows for storage of large quantities of various types of data;

- it allows the changing of data, addition of new data series, modification of tables (data structures);
- it gives the user simple and consistent view of the way data is stored in the system;
- it permits several users to select and access data according to many criteria, as it is impossible to specify in advance all the ways the data will be used;
- it allows for easy viewing of data, and contains facilities for validation of data;
- it provides facilities to interactively change data protection;
- it is able to store data about data (e.g., confidence levels);
- it provides a mechanism for assuring the integrity of the data; and
- it provides mechanisms for monitoring and tuning performance.

The modeling and analytical capabilities introduce several additional features. Since GMIS provides access to such faciliites as APL, PL/I, TSP, EPLAN, and FORTRAN, it provides the user with an efficient flexible environment to specify, construct, and execute statistical analyses and model studies, and to produce the associated plots and reports.

2. OVERVIEW OF THE SYSTEM ARCHITECTURE

Currently GMIS is implemented on an IBM System/370 computer. It uses the Virtual Machine (VM) concept extensively.[1] A virtual machine may be defined as a replica of a real computer system simulated by a combination of a Virtual Machine Monitor (VMM) software program and appropriate hardware support. For example, the VM/370 system enables a single IBM System/370 to appear functionally as though it were multiple independent System/370's (i.e., multiple "virtual machines"). Thus, a VMM can make one computer system function as though it were multiple, physically isolated systems.

A configuration of virtual machines used in GMIS is depicted in Figure 1, where each box denotes a separate virtual machine. Those virtual machines across the top of the figure are executing programs that provide user interfaces, whether they be analytical facilities, existing models, or data base systems. All these programs can access data managed by the general data management facility running on the virtual machine depicted in the center of the page. A sample use of this architecture might proceed as follows. A user activates a model, say in the APL/EPLAN machine. That model requests data from the general data base machine (called the Transaction Virtual Machine, or TVM), which responds by passing back the requested data. Note that all the analytical facilities and data base facilities may be incompatible with each other, in that they may run under different operating systems. The communications facility between

---

[1] The VM concept is presented in several places [Parmelee, 1972; Madnick and Donovan, 1974; and Goldberg, 1973], and many of its advantages are articulated elsewhere [Madnick, 1969; Buzen et. al., 1973]. The concept of "virtual machines" has been developed by IBM to the point of a production system release, VM/370 [IBM, 1972].

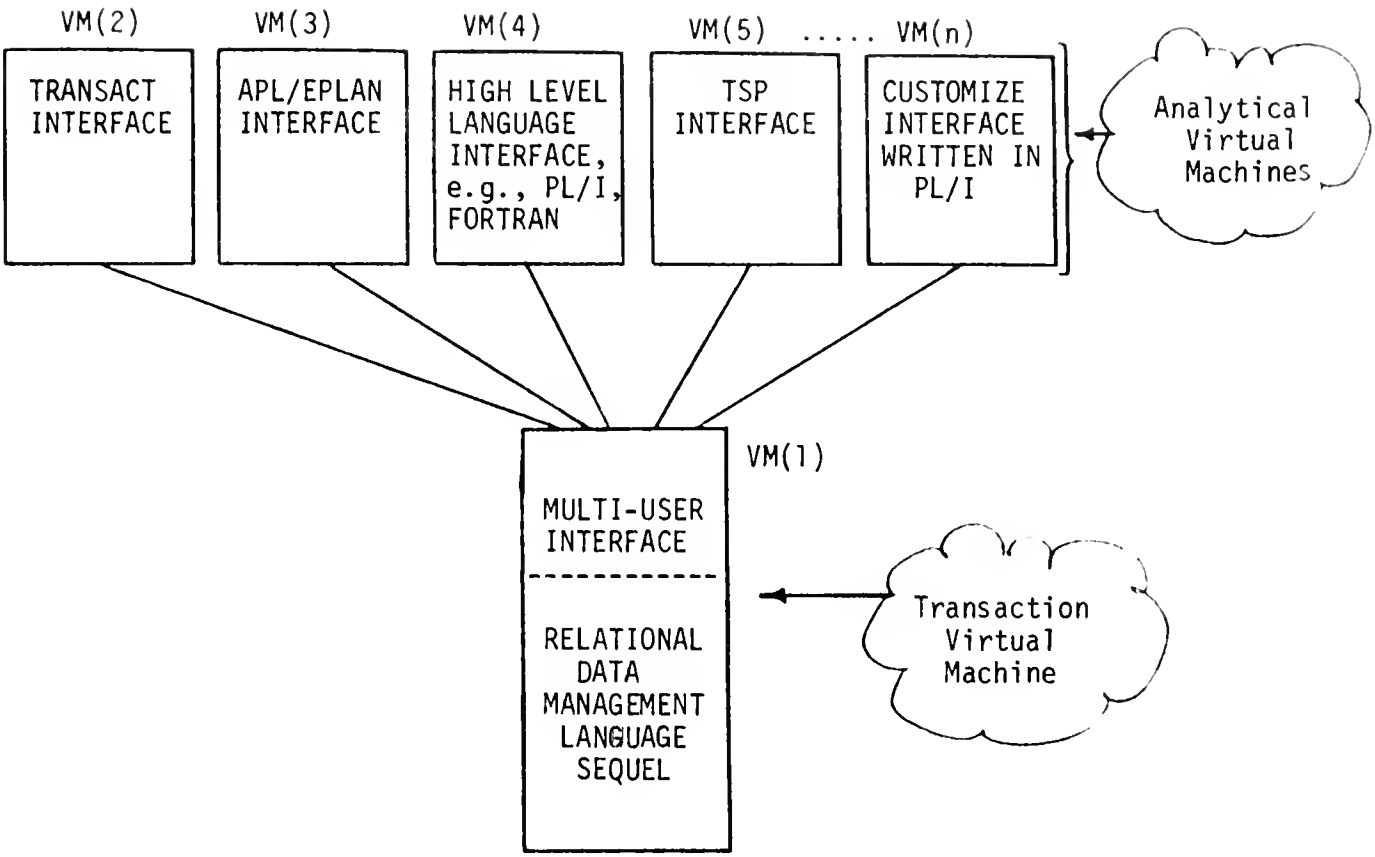

Figure 1: Overview of the Software Architecture of GMIS

virtual machines in GMIS is described in Section 4.1.1. Extensions to this architecture to allow interfaces to other data base systems and other computer systems are discussed in Section 4.1.2.

GMIS software has been designed using a hierarchical approach [Madnick, 1975, 1970; Dijkstra, 1968; Gutentag, 1975]. Several levels of software exist, where each level only calls the levels below it. Each higher level contains increasingly more general functions and requires less user sophistication for use. The transaction virtual machine depicted in Figure 1 shows only two of these levels, the Multi-User Interface and SEQUEL [Chamberlain, 1974]. The data base capabilities of this machine are based on the relational view of data [Codd, 1970]. In this section, each box will be briefly described. In Section 4 we return to describe some of the technologies used in implementing these boxes.

2.1 Structured English Query Language (SEQUEL)

We felt that the data management system would best be based on the relational model and hierarchical construction as this offered data independence, integrity, and a framework for reducing complexity. As part of our research on this topic, we proceeded with an implementation of an M.I.T. relational system [Smith, 1974]. However, in the current version of GMIS the data management capability is based on an experimental relational query and data definition language known as SEQUEL which has been developed at the IBM San Jose Research Laboratory [Chamberlain, 1974]. In cooperation with the IBM Cambridge Scientific Center and the IBM Research Laboratory at San Jose, we have extended this experimental system by easing restrictions on the data types it could

handle and relaxing constriants on the number of columns allowed in a table, and by increasing the allowable lengths of identifiers and character strings. We also designed mechanisms for security and for handling missing data, expanded the bulk loading facilities, added additional syntax, and made several changes to improve performance.

2.2 Multi-User Transaction Interface

Two requirements of GMIS are that multiple users be able to access the same data base and that different analytical and modeling facilities be able to access the data base all at the same time. For example, one user may want to build an econometric model using TSP while another user will request the system to generate a standard report. Still a third user may want to query the data base from an APL [Iverson, 1962; Pakin, 1972] environment. These requirements have been met with the design and implementation of the Multi-User Transaction Interface [Gutentag, 1975]. Each GMIS user operates in his own virtual machine with a copy of the user interface he requires. Each user transaction to the data base is written into a transaction file, and that user's request for processing is sent to the data base machine (Transaction Virtual Machine) as indicated in Figure 1. The Multi-User Interface processes each request in a first-in/first-out (FIFO) order, by reading the selected user's transaction file, and writing the results to a reply file that belongs to the user. Each user interface reads the reply file as if the reply had been passed directly from the data base management system. This procedure is discussed at greater length in Section 4.1.1 below.

2.3 User Interfaces

GMIS provides the capability for users to write their own interfaces to communicate with the data base system. TRANSACT is a general user interface that is designed to process transactions from most teletypewriters and CRT terminals. It allows the user to direct transaction output to any virtual device on the VM/370.

Interfaces to APL, TSP, EPLAN and PL/I are operational and enable users to communicate with the Transaction Virtual Machine (Figure 1) simultaneously with all other users. An interface to the TROLL econometric modeling facility is in the design stage.

The architecture depicted in Figure 1 also allows the use of any of these modeling or analytical facilities independent of the transaction virtual machine. For example, functions may be written in APL to operate on data stored in the APL's work space. TSP modeling and reporting capabilities can operate on data stored in TSP's data base. FORTRAN or PL/I can operate on data stored in the virtual machine that they are running. It should be noted, however, that not using the general data base facility seriously inhibits flexibility and makes the algorithms dependent on the physical organization of the data but more importantly inhibits the community of users as they cannot conveniently access the common data base.

3. SAMPLE APPLICATION OF GMIS

To demonstrate the characteristics of the existing GMIS System, we use an example drawn from work done for the Federal Energy Administration on the construction of indicators of domestic energy conditions [M.I.T. Energy Laboratory, 1975].[1] The object of this particular indicator was to give a picture of future trends in gasoline consumption. It was proposed that the indicator be depicted as a series or plot of the average miles per gallon of each month's new car sales. Policymakers could note if the average fuel efficiency of new cars was going down or up, hence reducing or increasing future demand for gasoline.

The indicator is shown in Figure 2. Several points concerning the figure and its derivation are worth noting:

(1) The plot covers the 15-month period from January 1974 to March 1975. It is surprising to find that during the "energy crisis" the average miles per gallon of new cars sold actually went down! We had initially expected that during that time people would have purchased smaller, more efficient cars, resulting in an increase in average miles per gallon. Why did it go down?

(2) Note that since the graph raises additional questions, it becomes necessary, in order to resolves these questions, to access and analyze the data in ways not originally planned for.

---

[1] Marvin Essrig is responsible for the initial construction of this example.

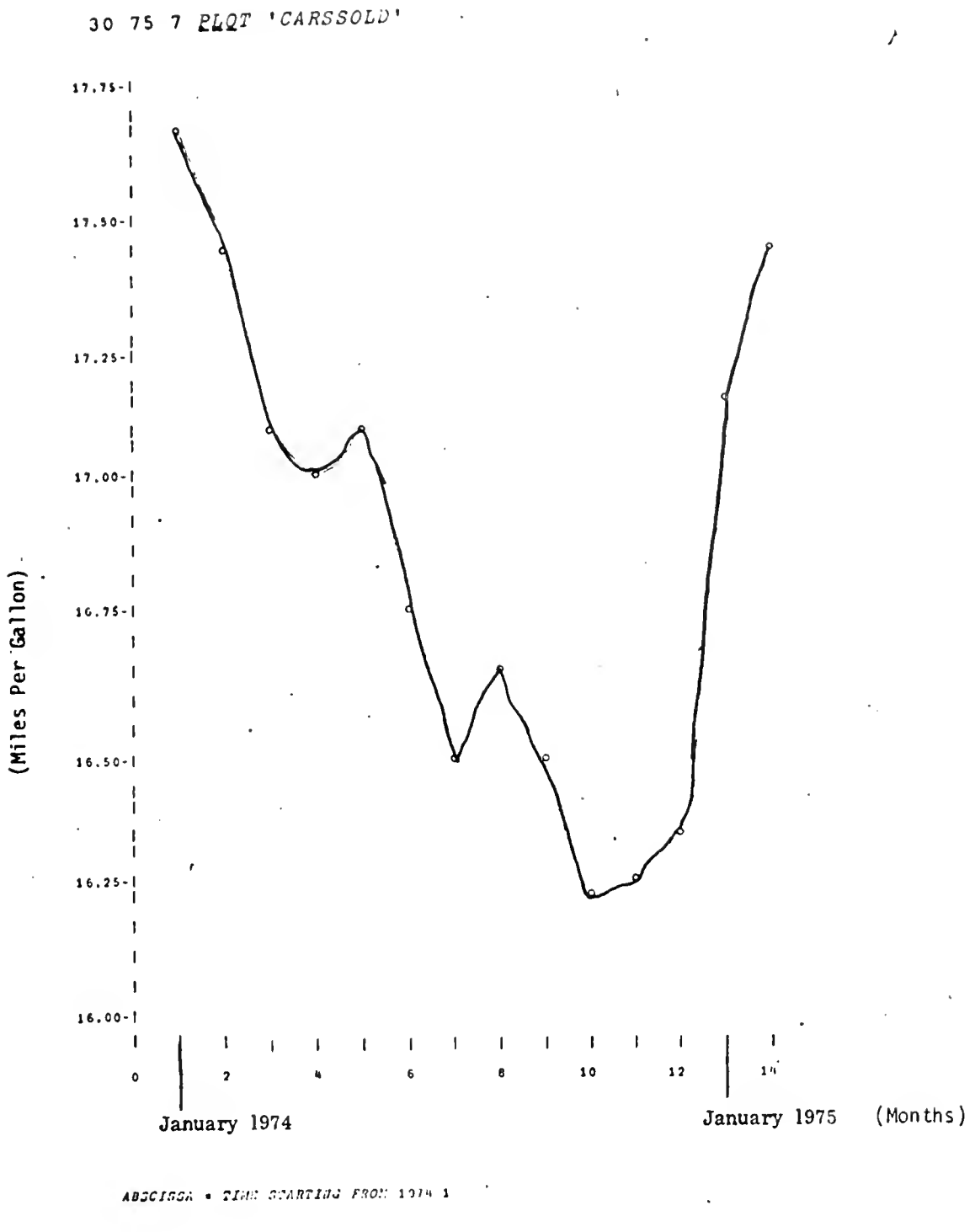

Figure 2. Average Miles per Gallon of Cars Sold in a Month

(3) The data from which the graph is derived comes from a variety of sources, each using different terminology and dissimilar means of presentation.

(4) The data is both numeric and non-numeric (e.g., name of models of cars).

The remainder of this section shows how GMIS was used to construct and analyze this indicator. Two user interfaces of GMIS will be used:

(1) TRANSACT is an interface to the data management level (SEQUEL), which includes a Data Definition Language (DDL) and Data Manipulation Language (DML). This level can be used to:

- restructure the data,
- input the data, and
- query data.

(2) APL/EPLAN is the anlaytical, modeling, and statistical level, which resides above the multi-user interface (Figure 1). EPLAN is a set of routines imbedded in APL for doing statistical functions and reporting.[1]

3.1 Data Manipulation

An example of creating a table and inserting data into it via TRANSACT-SEQUEL will demonstrate how a user stores data in GMIS. Note that all data are viewed as residing in tables, as in the relational model of data [Codd, 1972]. The tables have columns whose entries come from sets of elements called domains. Figure 3 is an example of a table.

---

[1] EPLAN is now available as an IBM product under the name "APL Econometric Planning Language" [IBM, 1975].

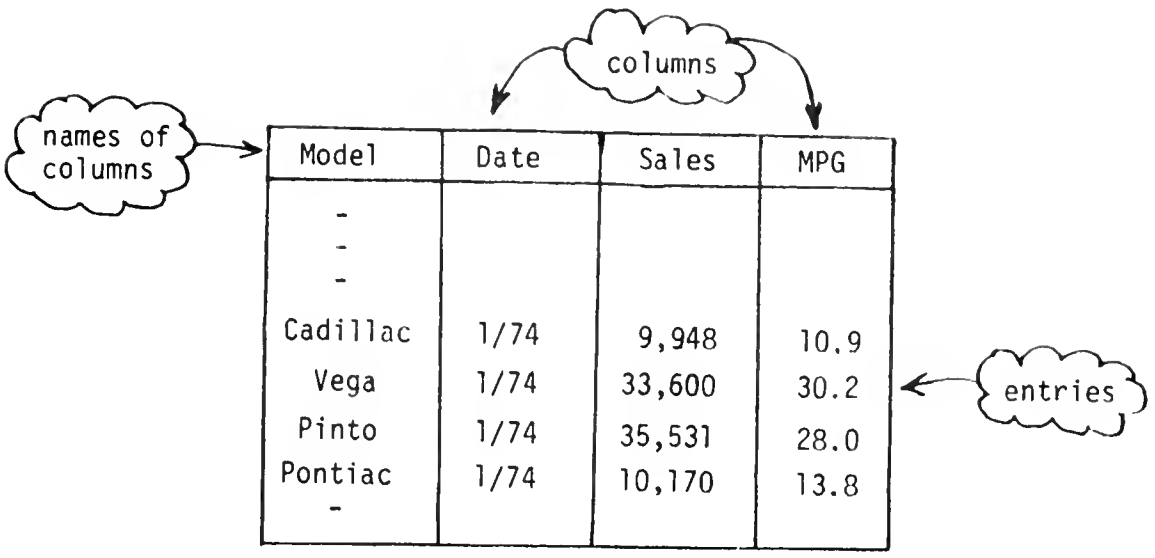

Figure 3. Sample Table

3.1.1 Data Definition Facility

A data structure is created in TRANSACT by using SEQUEL commands[1] by first defining the desired domains, then declaring a group of columns to be a table, and finally, inserting data into the table.

The interactive session to create the table presented in Figure 3 is found in Figure 4, where the commands shaded are user inputted. The first four commands establish the existence of the four domains: (models), (vol), (date), and (mpg). The domain 'model' will hold information stored as characters, while 'date', 'vol', and 'mpg' will consist of numeric data.

---

[1] A complete syntax description of TRANSACT and SEQUEL commands is available in a GMIS Primer [M.I.T. Energy Laboratory, 1975].

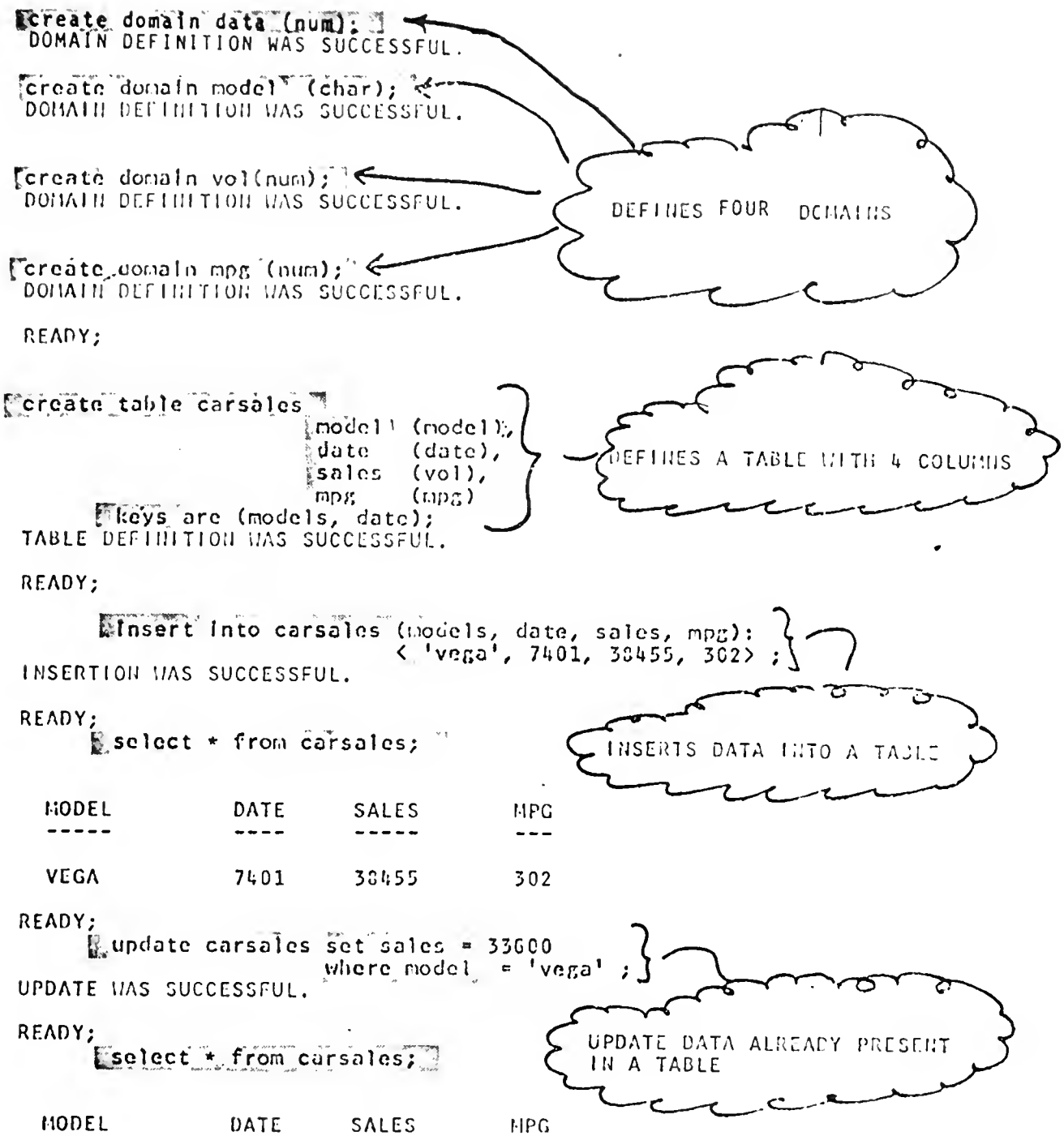

Figure 4. Example of Table Creation and Data Entry

The next command creates a table called CARSALES. The first column is labelled MODEL, and entries in this column will be classified as belonging to the set (or domain) model. The other three columns are defined in a similar fashion, where entries in the column sales are the volume of cars sold during the month entered in the column date.

The INSERT statement of Figure 4 results in the insertion of one entry into each column of table CARSALES. The SELECT * command results in the printing of all entries in table CARSALES. The UPDATE command results in changing one entry in the table. Note that the change is reflected in the output from the next SELECT command.

3.1.2 Bulk Loading Facility.

Suppose that a great deal of data were to be loaded into table CARSALES. Inputting it via the console, as in the previous example, would be prohibitively slow and costly. A bulk loading facility has been implemented to reconcile this matter. A series of data cards and their appropriate header cards for input into the bulk loader are shown in Figure 5. The bulk loader will accept these cards, define the indicated domains, create the table, and insert the data into the appropriate columns of the table. For a complete explanation of formats and uses of the bulk loader, see the "GMIS Primer" [M.I.T. Energy Laboratory, 1975].

carsales data

```
$DEFDOM   MODEL      CHAR
$DEFDOM   VOL        NUM
$DEFDOM   MPG        NUM
$DEFDOM   DATE       NUM
$DEFTAB   CARSALES
          MODEL      MODEL
          DATE       DATE
          VOLUME     VOL
          MPG        MPG
$PRIKEY   MODEL      DATE
$LOADTAB  CARSALES
          MODEL      1          1          1          15
          DATE       1          20         1          23
          VOLUME     1          28         1          34
          MPG        1          17         1          19
$ENDCOL
CHEVROLET          1247401      33108
CORVETTE           1547401       2078
CHEVELLE           1797401      21175
CHEVY NOVA         1877401      21464
SPORTVAN           1527401       1370
MONTE CARLO        1497401      15668
CAMARO             1797401       8787
VEGA               3027401      38455
PONTIAC            1387401      10170
GRAND PRIX         1037401       4042
FIREBIRD           1797401       3666
VENTURA            1217401       4890
OLDSMOBILE         1107401      10533

$ENDLOAD
$ENDINP
```

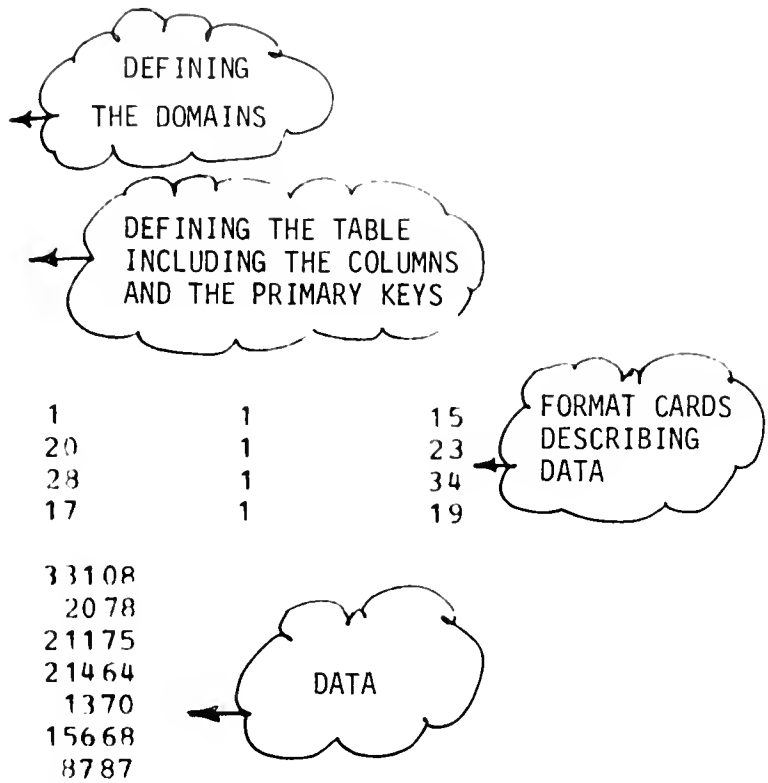

Figure 5. Example of a File Ready for Bulk Loading

3.1.3 System Inquiry Facility

The TRANSACT-SEQUEL level has a number of "system commands" for inquiring about tables as opposed to their contents. For example, Figure 6 demonstrates some of these commands. The first command lists all tables that have been created. Note that the system created three tables (INTEGRTY, DOMCAT, and CATALOG) for its own use. The next command lists information about the table, CARSALES, where the system response lists the name of each column, the domain from which the entries for that column are taken, and the data type of each column (either "CHAR" or "NUM"). The next command lists information about domains.

3.1.4 Query Facility

Figure 7 illustrates queries to the tables that have been created. All queries start with the word SELECT. The first two queries ask the system to list the contents of the tables CARSALES and MILEAGE. The rest of the queries contain a "WHERE" clause which allows the user to select only data that meet certain requirements. Note that the SELECT command may be used to specify queries that require data from more than one table. The general form and syntax of the SELECT command is found in the "GMIS Primer" [M.I.T. Energy Laboratory, 1975].

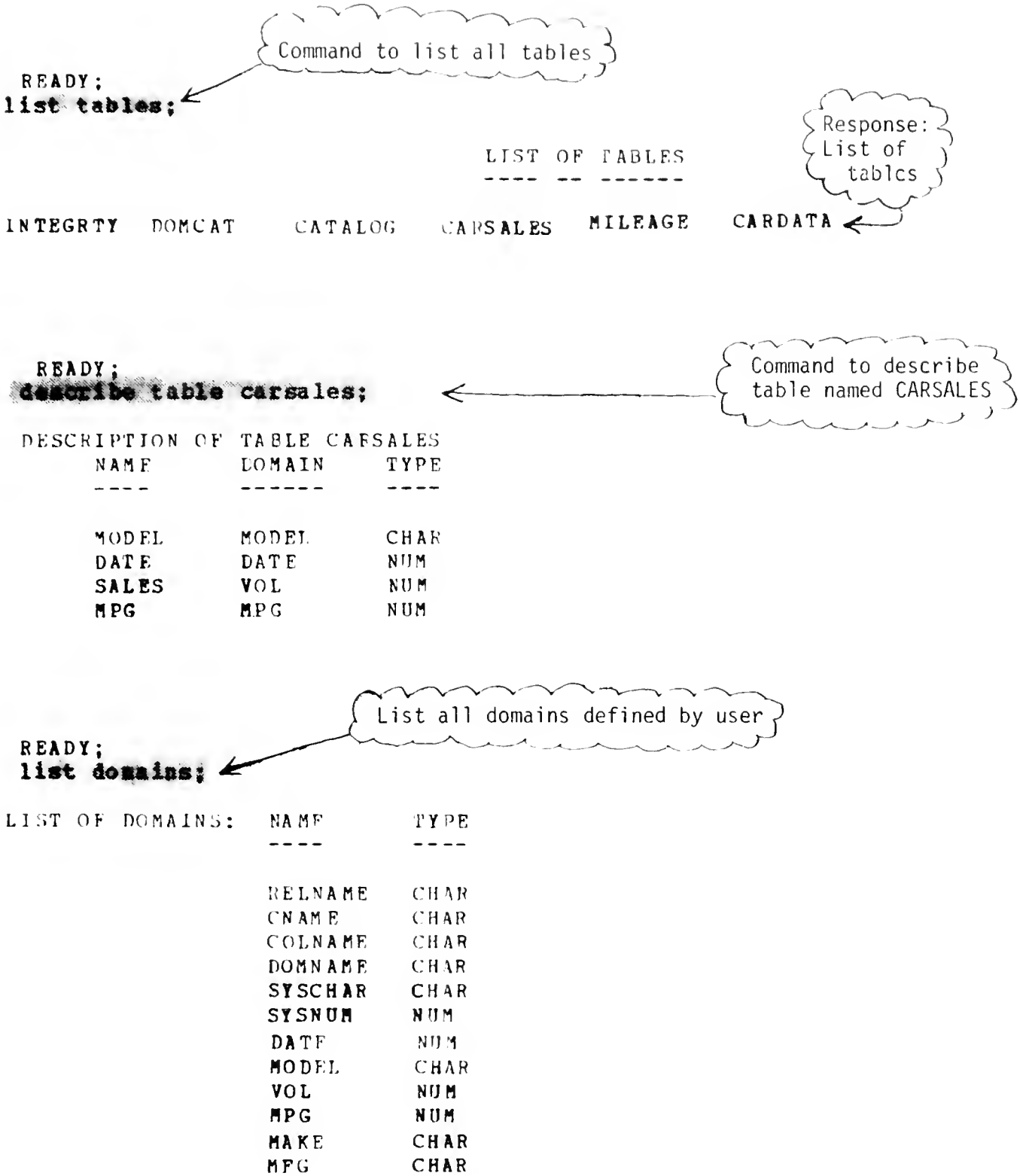

Figure 6. Examples of Inquiries about Tables

```
select * from carsales;
```

Display all entries in table named CARSALES

MODEL	DATE	SALES	MPG
VEGA	7401	33600	30.2
CHEVROLET	7401	33108	12.4
CORVETTE	7401	2078	15.4
CHEVELLE	7401	21175	17.9
CHEVY NOVA	7401	21464	18.7
SPORTVAN	7401	1370	15.2
MONTE CARLO	7401	15668	14.9
CAMARO	7401	8787	17.9
PONTIAC	7401	10170	13.8
GRAND PRIX	7401	4042	103

Entries in table CARSALES

.
.
.

```
READY;
select * from mileage;
```

MODEL	YEAR	MPGCITY	MPGHWY	MPGAVG
GREMLIN	1975	19	24	210
HORNET	1975	18	24	203
MATADOR	1975	14	19	159
APOLLO	1975	16	21	179
SKYHAWK	1975	19	25	213
CENTURY	1975	16	24	188
LESABRE	1975	12	16	135
ELECTRA	1975	11	15	125

.
.
.

Figure 7. Sample Table Queries

```
READY;
select model,mpgavg from mileage where mpgavg between 20.0 and 30.0;
```

MODEL	MPGAVG
GREMLIN	210
HORNET	203
SKYHAWK	213
MONZA	222
PINTO	209
MUSTANG	209
STARFIRE	213
VALIANT	200
ASTRE	222
MUSTANG	247
PINTO	280

List all models where average mileage is between 20.0 and 30.0

```
READY;
select avg(mpgavg) from mileage
where mpgavg>0;
THE RESULT OF YOUR QUERY IS:
          15.5
```

AVG OF MPG IS 15.5

Select the highest MPG

```
READY;
select model,year,mpgavg
from mileage
where mpgavg =
        select max(mpgavg)
        from mileage;;
```

MODEL	YEAR	MPGAVG
VEGA	1974	30.2

THE HIGHEST MPG IS A VEGA

Figure 7 (cont'd). Sample Table Queries

3.2 Modeling and Analytic Functions

3.2.1. Validating Data. The data for this example indicator came from many sources. Data in the table CARSALES came from "Ward's Automobile Reports" [Ward's, 1975]. The data in the table MILEAGE came from two Environmental Protection Agency documents [EPA, 1974; EPA, 1975]; 1974 data was found in the "1974 Gas Mileage Guide for New Car Buyers," and the 1975 data was from a similar document entitled "1975 Gas Mileage Guide for New Car Buyers."

The data stored in the MILEAGE table was entered (using the bulk loading facility) as it appeared in the 1974 and 1975 EPA documents. However, inconsistencies resulted from two factors:

(1) Miles per gallon (mpg) for 1974 data was a single number averaging city and highway driving, whereas data for 1975 was two numbers reflecting both city and highway driving.

(2) There was a 5% change in the method used by the EPA to determine the mileage values from 1974 to 1975.

Let us demonstrate the interaction between a modeling facility and the data base facility by normalizing the data to reflect the inconsistency in (1) above, thus allowing fair comparison between 1974 and 1975 mpg data. We perform the following three steps using the APL level: (The reader should keep in mind Figure 1, depicting the relationship between the two virtual machines, one running APL and the Transaction Virtual Machine).

(1) Extract data from the data base facility.

(2) Perform a correcting function on it.

(3) Insert the corrected data back into the data base facility.

Figure 8 exhibits the console session to perform the above three tasks. Our strategy is to convert for each model the two 1975 numbers (mpg in city driving, mpg on the highway) into one comparable to the one 1974 number.

(1) To extract the data (city mpg, highway mpg for each model for 1975) we use the QUERY command of Figure 8. The QUERY command is a function that has been added to APL to interface between the two virtual machines. The APL QUERY function passes the given SEQUEL command in quotes to the Transaction Virtual Machine. The TVM then gets the data and passes it back to the APL workspace and APL prints the names of the vectors passed back, in this case MODEL, CITYMPG, and HWYMPG. The software mechanisms for accomplishing this communication are transparent to a user at the APL level. They are described later in Section 4.1.1.

(2) The following function was performed on city miles per gallon, and highway miles per gallon to get one value that was consistent with 1974 values.

$$\text{Avg.MPG} = \frac{1}{\frac{.45}{\text{HWYMPG}} + \frac{.55}{\text{CITYMPG}}} \qquad (1)$$

In Figure 8 function (1) was envoked by typing its name, 'CHANGE'. For the reader's information we listed the APL implementation of function (1). Note that the APL implementation not only performed function (1), but it also created the necessary QUERY command to insert the new data back into the data base.

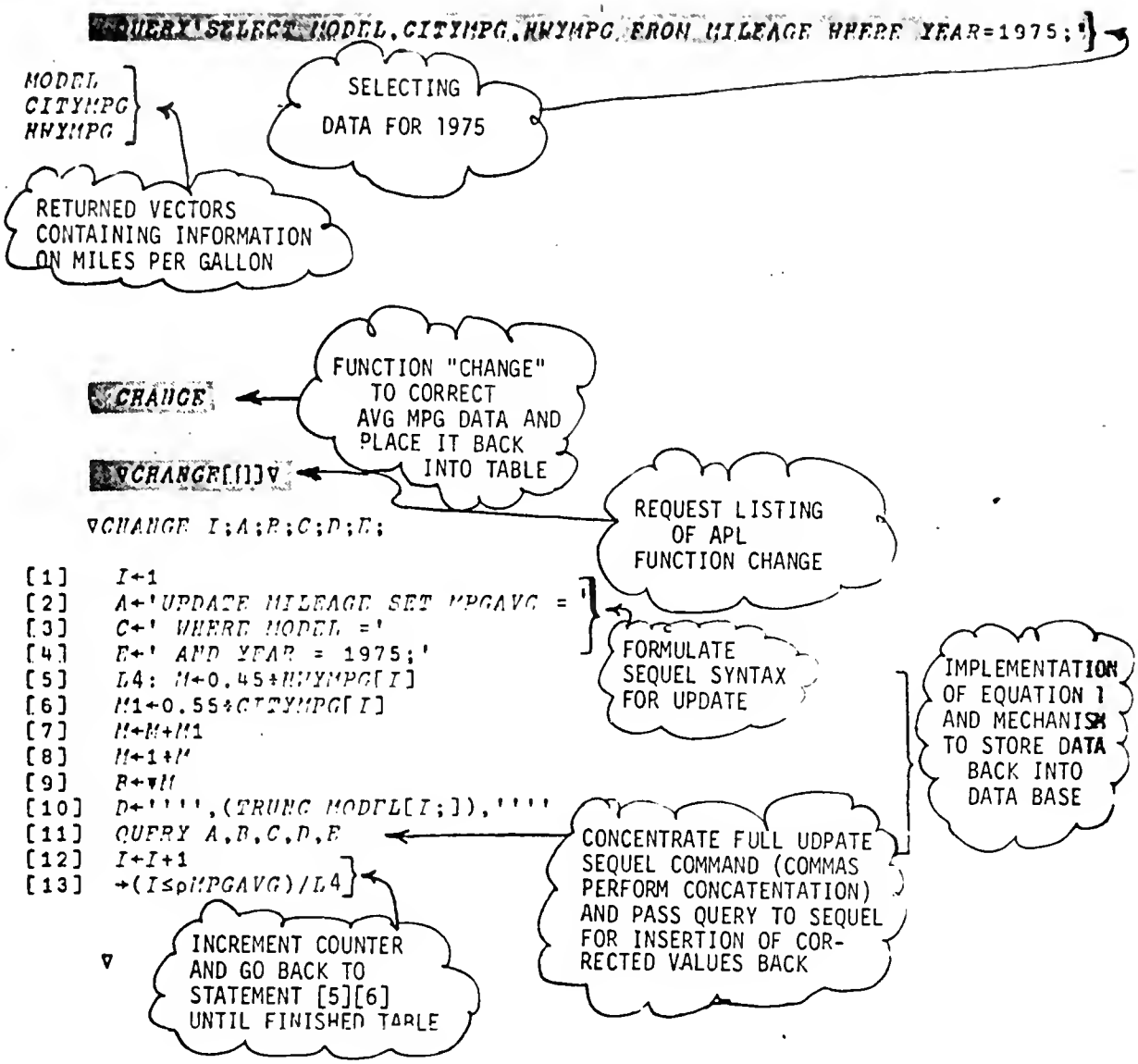

Figure 8: Example Cleaning of Data

The reader who is not familiar with APL can use the comments of the listing. It is not necessary for readers of this paper to thoroughly understand APL. For those who wish to do so, the references [Iverson, 1962; Pakin, 1972] can be consulted.

A similar function was applied to correct the 5% difference in data reporting of (b) above.

3.2.2 Reporting

A GMIS user has the full reporting capabilities of any of the modeling or analytical facilities at his disposal. For example, a GMIS user can employ the APL/EPLAN facility as a report generator and to produce plots. To produce the indicator plotted in Figure 2, the following steps were followed.

(1) Use the QUERY command to extract the desired data

(2) Execute an / APL function to calculate the average miles per gallon of all cars sold during a given month from the data in the three created tables using the following formula:

$$\text{Average Mpg. All Cars} = \frac{\sum_i \text{Vol}_i \times \text{Mpg}_i}{\sum_j \text{Vol}_j}$$

(3) Convert the resulting vector into a time series.

(4) Use the EPLAN plot facility to produce the PLOT of Figure 2.

As was discussed in Section 3, this plot raises several questions. Why did the average miles per gallon of all cars sold during the months of the energy crisis go down? We had expected that it would go up because people would have bought high mileage cars during a shortage of gasoline.

One possible explanation is that the wealthy were relatively unaffected by the energy crisis and thus they continued to buy large, luxurious, lower mileage cars. This may have resulted in a disproportionate smaller number of compact, low-mileage cars sold. Another explanation might be that the car dealers, seeing an end to the popularity of large cars, lowered prices on these models greatly, thus inducing a larger than expected sale of these cars. Another is that foreign compacts (which we did not include) encroached on the sale of American compacts.

In order to resolve these questions, it becomes necessary to access the data in a different way than we had initially expected. A plot of the sales of a luxury car (e.g., Cadillacs) and the sales of a compact (e.g., Valiants) over the same period would indicate how the sales of these groups behaved during that period.

Again, operating on the modeling level, the following three steps are taken (the corresponding console session is shown in Figure 9).

(1) Extract the data using QUERY commands

(2) Convert the data from a vector to a time series using the APL DF function[1], e.g.,

```
CADILLAC_SALES ← DF.VOLUME
```

(3) Use the EPLAN P L O T function to produce the desired plot.

---

[1] In APL all function names, such as DF and PLOT, are underlined, as can be seen in Fgiure 9. Since variable names cannot have spaces in them, underscores also are commonly used to clarify variable names, as has been done with CADILLAC_SALES in the figure.

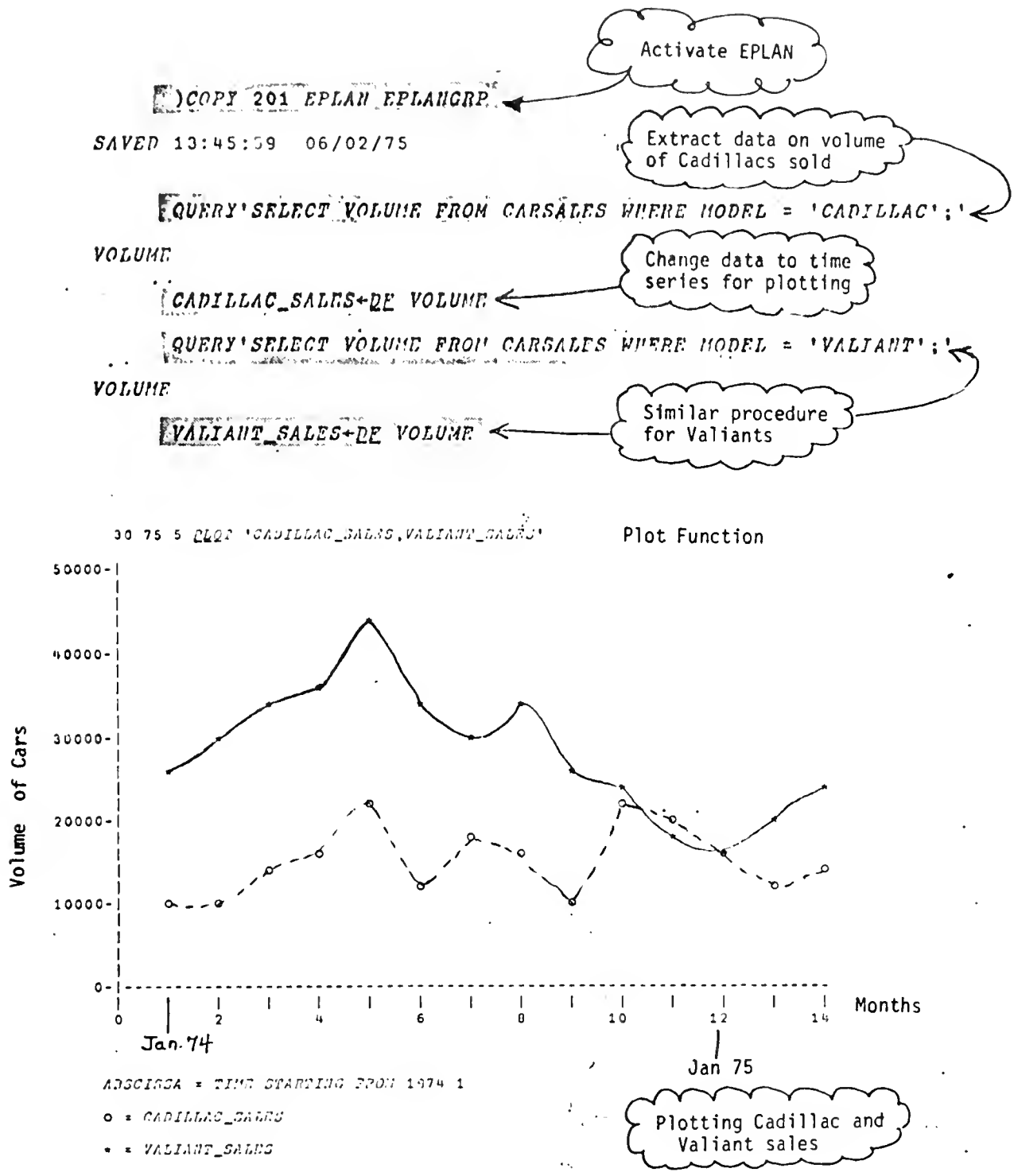

Figure 9. Using the Plotting Function for Reporting Data

Note that the plot has car sales on the vertical axis and months on the horizontal axis. The 'o' denotes Cadillac sales, the '*' denotes Valiant sales. Figure 9 reveals that the sales of Valiants showed a definite downward trend starting from about the fifth month of 1974, while the sale of Cadillacs remained relatively constant.

3.2.3 Modeling

In recent years increasing emphasis has been placed on the use of models to aid in policy decision making. A model is roughly defined as an incomplete representation of a system, where the purpose of the model governs which elements of a model can be adjusted to simulate a real world change in policy. The results of the simulation can then be studied and compared with other simulated courses of action before a final decision to effect change in the actual system is made.

Another useful feature of a model is that it serves as a facility through which relationships between elements of a system can be explored. We can illustrate this capability by performing a simple analysis of the data already introduced in this example. Suppose one wanted

to investigate the mathematical relationship between average miles per gallon of all cars sold in a month with that of all cars sold in some previous month. A correlation matrix depicting the strength of the relationship between average miles per gallon of all cars sold in a month with that of the previous month, and with that of two and three months ago, gives an insight into how a mathematical model of this relationship might behave. The EPLAN C O R and L A G functions have been applied to the available data resulting in the correlation matrix show in Figure 10.

	MPG_t	MPG_{t-1}	MPG_{t-2}	MPG_{t-3}
MPG_t	1			
MPG_{t-1}	.62	1		
MPG_{t-2}	-.04	.54	1	
MPG_{t-3}	-.09	.39	.88	1

Figure 10: Correlation Matrix

Inspection of Figure 10 reveals that one ought to expect that the average miles per gallon of all cars sold in a month is somehow strongly related to the average miles per gallon of all cars sold in the previous month, but does not appear to be highly correlated with the figures from two or three months ago (a correlation coefficient close to ± 1 is regarded as an indication of a strong relationship between two variables, whereas a value of 0 indicates a weak relationship). To explore this relationship further, an ordinary least squares regression analysis is applied to the two variables using the EPLAN R E G function (Figure 11). More precisely, we seek an equation of the form:

$$\text{AVG MPG of CARSSOLD}_t = \alpha_0 + \alpha_1 \cdot \text{AVG MPG of CARSSOLD}_{t-1}$$

The estimated values of the coefficients α_0 and α_1 from the table in Figure 11 are 4.928 and 0.706, with standard errors of 3.2 and 0.2, respectively. The fourth column of the figure depicts the T statistic for the estimated values of α_0 and α_1.

```
      'CARSSOLD' REGRESS '1,(1 LAG CARSSOLD)'
WITH:

COEF/VALUE/ST ERR/T-STAT.....

 1    4.92827   3.19856   1.54078
 2    0.70615   0.19011   3.71443

NO OF VARIABLES.........    1.00000
NO OF OBSERVATIONS......   13.00000
SS DUE TO REGRESSION....    1.29413
SS DUE TO RESIDUALS.....    1.03177
F-STATISTIC.............   13.79701
STANDARD ERROR..........    0.30626
R*2 -STATISTIC..........    0.55640
R*2 CORRECTED...........    0.55640
DURBIN WATSON STATISTIC.    1.10338
CARSSOLD← ( 4.928          T 1 ) + ( 0.706 T (1 LAG CARSSOLD))
```

Figure 11: Sample Regression

Based on the results of this initial exploration, more complex formulations may be devised to help explain the behavior of the sales of different car models over this period, and all would be constructed in the manner shown in Figure 11. Moreover, once underlying behavioral relations had been estimated, it might be desirable to build a simulation model to forecast

automobile fuel consumption in the future. Once again, all the programming tools and higher-order simulation languages could be made available through the system outlined in Figure 1, with access to all the data and estimated relations produced in the course of the analysis.

4. DETAILS OF THE GMIS DESIGN

There are three basic features of the GMIS system that give it its flexibility: (1) an overall system architecture making use of the (largely untapped) power of VM, (2) construction of the system within a hierarchical framework, and (3) the use of a relational representation of data. Section 2 gave a brief introduction to these features, and here we discuss the role of each in greater detail.

4.1 The Use of VM in the Software Architecture

Through the use of the VM concepts and the proposed architecture of Figure 1, a number of the important features of GMIS become possible, or much easier to implement:

(1) Multi-user coordination of access and update to a central data base.

(2) An environment where several different modeling facilities can access the same data base.

(3) An environment where several different and potentially incompatible data management systems can all be accessed by the same user models or facilities.

(4) Increased security and reliability [Donovan and Madnick, 1975].

VM also has disadvantages, the primary one one being the potential increase in overhead costs associated with the synchronization and scheduling of the VM system.

Figure 1 depicts a configuration of virtual machines operating on a single real computer. At the present time PL/I, FORTRAN, EPLAN/APL, and TSP are the only facilities interfaced with the data management system. Work is under way to bring TROLL to this status. Some of these modules operate under a different operating system but are made to run on the same physical machine using VM/370. All the modeling or analytic virtual machines may request data from the general data management system. In this section we discuss the techniques we used to facilitate the communications between these virtual machines, performance analysis, and proposed extensions to this architecture.

4.1.1 Communication between VM's

As part of the IBM/MIT Joint Study a multi-user interface on the data base machine has been implemented [Gutentag, 1975]. This interface allows several users (programs running on the VM's) to access the single data base system. Note that for this section a distinction is made between a human user and a "user" of the multi-user interface, which is usually another program.

Essentially what is needed is a means of passing commands and data to the data base machine, returning data, and a locking and queueing mechanism. One way to pass data is to use virtual card readers and card punchers. The data base virtual machine would be in wait state trying to read a card from its virtual card reader, the analytical machine would punch the commands on the virtual card reader that would be read by the data base VM. This mechanism is inefficient, however, and does not allow flexible processing algorithms.

The mechanism implemented in GMIS is as follows (note that this mechanism is invisible to a modeler when he envokes the APL/EPLAN level command QUERY, as this command automatically envokes the mechanism). Each user virtual machine (UVM), which is accessed by logging on to a separate account ID under VM/370, sends transactions to the Transaction Virtual Machine through a communications facility (described below). The Multi-User Interface (MUI) stacks these transaction requests and processes them one at a time. The results of each transaction are passed back to the virtual machine that made the request through the same communications facility. Replies to the transactions may be processed with any software interface that is required for the application. The APL/EPLAN interface discussed earlier has been implemented in this manner.

The best way to explain how the MUI works is to follow a user's virtual machine's transaction through each processing step. Refer to Figure 12 for an illustration of the transaction processing scheme described below. Each user virtual machine must have a small virtual minidisk attached to it that has been supplied with a multi-write password. This password allows more than one virtual machine to link to the disk with read/write privileges (otherwise, VM/370 only allows one user at a time to link to a disk with writing privileges).

When a user's virtual machine wants to send a transaction to the data base, it writes the transaction onto its multi-write disk in a CMS[1] file that is reserved for transactions (steps 1 and 2 of Figure 12). The user's

---

[1] CMS [IBM, 1974] is an operating system commonly run under VM/370.

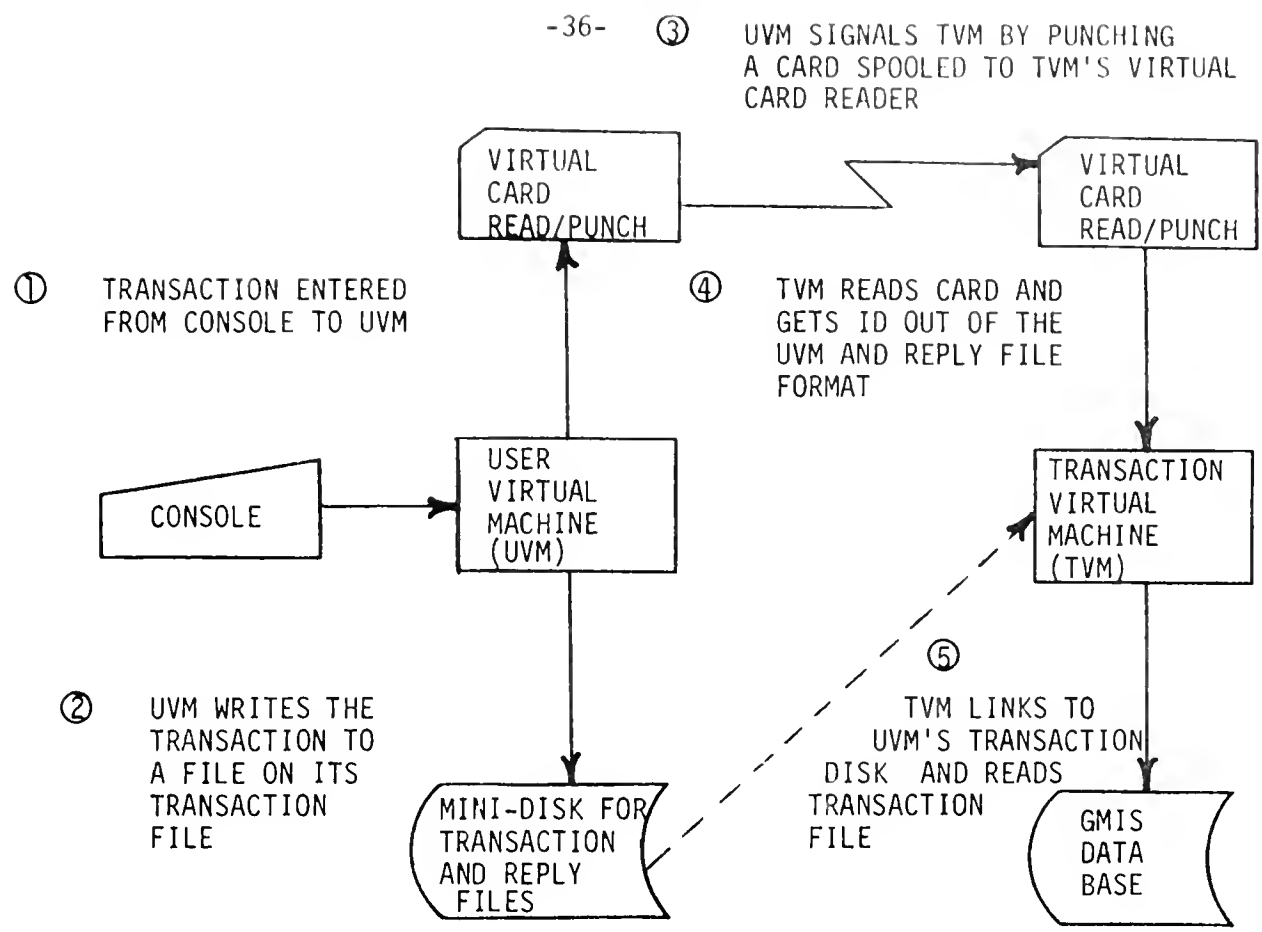

Figure 12a. Sending a Transaction Request

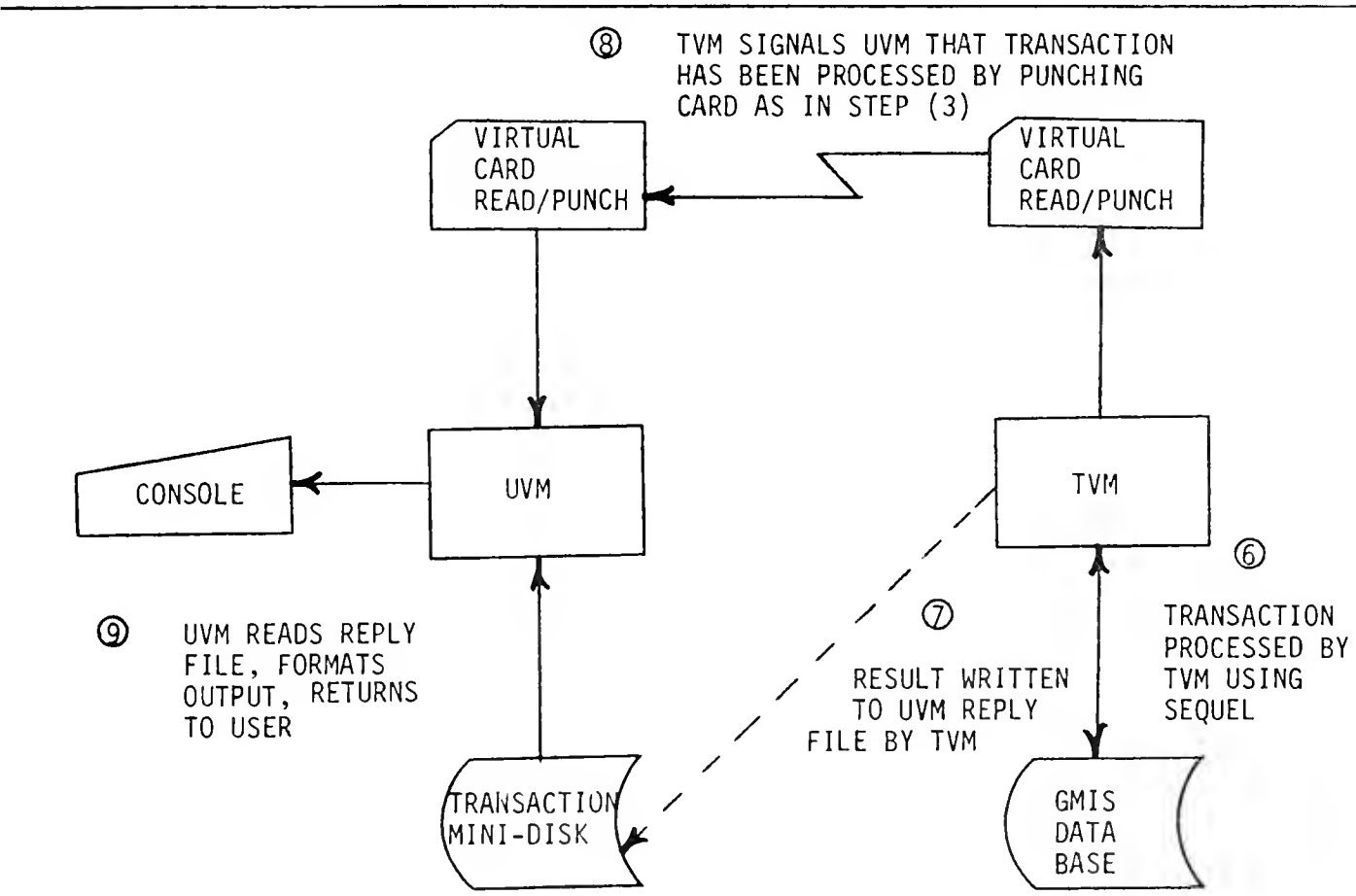

Figure 12b. Returning Data

virtual machine must then signal to the MUI that it wants its transaction to be processed. This is done by directing the VM/370 Control Program (CP) to send all output from the user's virtual card punch to the virtual card reader of the Transaction Virtual Machine (TVM). The user's virtual machine then punches a single virtual card containing two items of information: the ID of his virtual machine, and a code indicating the type of file format that the MUI must use when passing the transaction reply back to the user virtual machine (step 3).

Each card punched by a user is actually a request to the MUI to process a transaction residing in the user's transaction file. These cards are stacked in the card reader of the TVM, and are processed one at a time, where the first card stacked is the first to be processed (FIFO) (step 4).

The MUI is always running in a wait state or processing transactions. When a card is received by the TVM's virtual card reader, an interrupt is generated that activates the MUI to begin reading from its card reader.

To read the user's transaction, the MUI must first access the user's transaction file. This is done by first linking to the multi-write disk of the virtual machine given by the ID on the transaction request card. (The multi-write disk is always attached at the same virtual address; in the current implementation, disk address 340 is used for all transaction files.) The disk is then accessed by the MUI, and its SEQSTAT SEQUEL file is read (step 5). It should be noted that the SEQUEL software level provides a file reading capability.

After the transaction has been processed by SEQUEL in the usual manner (step 6), the MUI writes this reply on the user's multi-write disk in a file called SEQUEL REPLY (step 7). One of several file formats may be used, depending on the user's software environment. Three general formats have been proposed that will satisfy all currently anticipated GMIS requirements. One format is to be read by APL programs, another format will be compatible with TROLL files, and a third format will be compatible with any language that can process sequential CMS files (e.g., PL/I, FORTRAN). The user's transaction request card indicates which file format is to be used by the MUI.

The TVM then punches a virtual card to the UVM to signal completion of transaction processing (step 8). Finally, the UVM reads its SEQUEL REPLY file, and processes the transaction result in its own environment (step 9).

4.1.2 Extensions of Architecture

The following three extensions to the architecture of Figure 1 merit further investigation.

Incompatible Data Systems. Figure 13 depicts an extension of the architecture that would allow different and perhaps incompatible data base systems to be accessed by the modeling facilities. The general data base system would act as a catalog for data stored in the decentralized system. The data management virtual machine acts as an interface, analyzing the data query and funneling it to the appropriate data base management system. These mechanisms could be made invisible to the user, who can use the system as though he had all the data in one "virtual" data base. The implication of this extension on synchronization, data updating, and performance must be further researched.

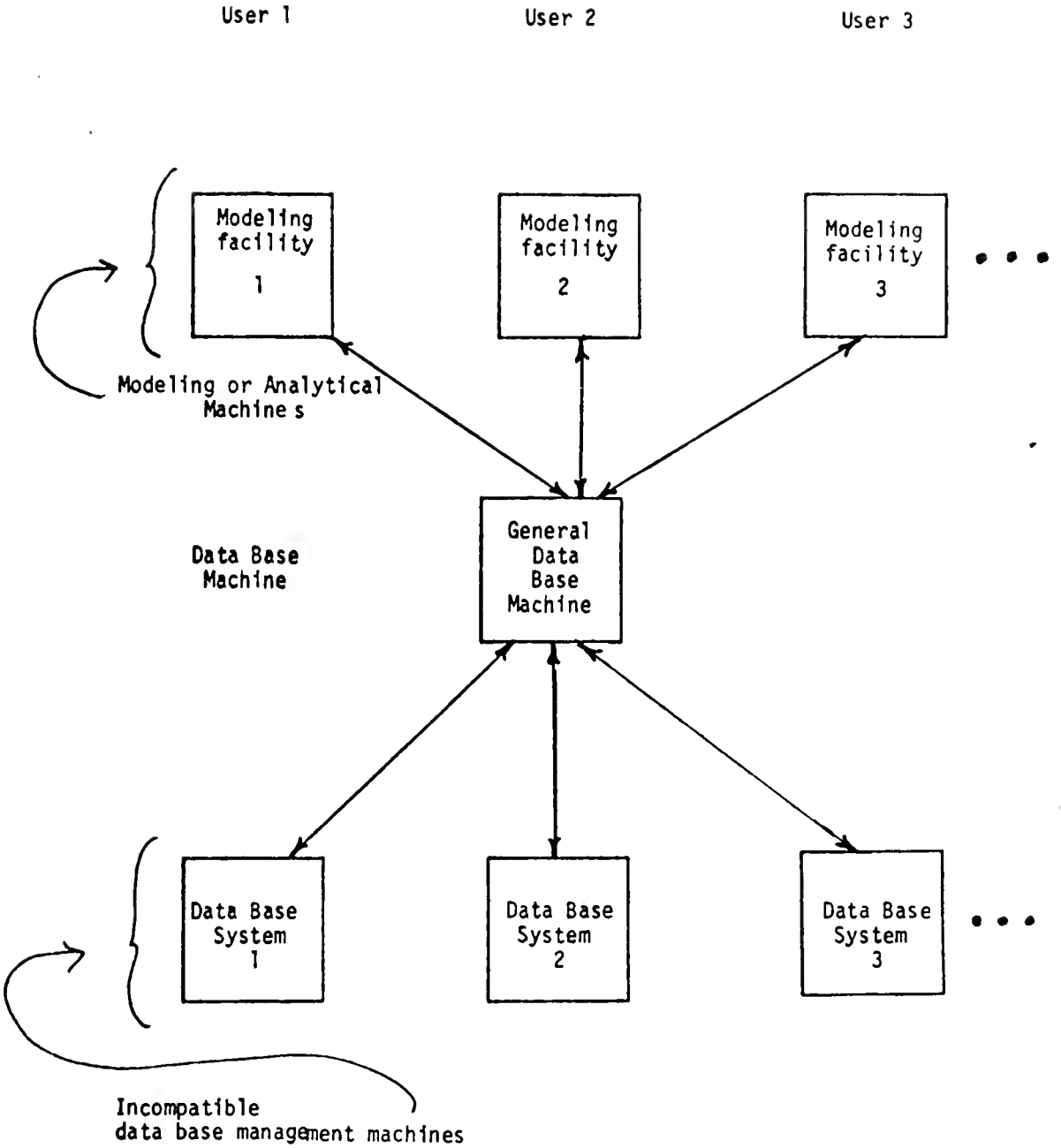

Figure 13: External Architecture

Standardization of data base systems. It may be useful to place user interfaces that are syntactically and semantically equivalent to existing data management systems (e.g., IMS, TOTAL) above the general data base system of Figure 1. This would allow data to be inputted and validated in a data system with which a user is familiar, and then stored in a standardized general data base system.

Decentralized/centralized data bases. The advantages of decentralized data bases are that they are usually maintained by the people that are using them. The advantage of a centralized data base is that many groups of people can access it. The above architecture may be extended to interface not only with data base and modeling systems running in other virtual machines, but to other remote computers /including non-IBM equipment. The implication of this extension on data updating and networking problems must be investigated with further research.

4.1.3 Degradation of Variable Cost with Multiple VM Operation

The construction of a system of communicating VM's brings great advantages, but these come at the expense of some sacrifice in performance.[1] Various performance studies of VM's are available in the literature [Hatfield, 1972, Goldberg, 1974], and we are engaged in a theoretical and empirical analysis of the degradation of variable cost performance as a function of the number of modeling machines [Donovan, 1975]. The direction of this work can be seen by considering a configuration as in Figure 1, where several modeling facilities, each running on a separate virtual machine, are accessing and updating a data base that is managed by a data base management system running on its separate virtual machine. What is the degradation of performance with each additional user? What determines the length of time the data-base machine takes to process a request? What is the best locking strategy?

An access or update to the data-base machine may be initiated either by a user query, which would be passed on by the modeling machine, or by a model executing on the modeling machine. In either case, the data-base machine while processing a request locks out (queues) all other requests. The analysis is further complicated by the fact that as some VM's become locked, then others get more of the real CPU's time, and therefore generate requests faster. However, the data-base VM gets more of

---

[1] Here we are addressing the issue of variable costs. Later in Section 5.2 we address the more important issue, fixed costs, for applications like those addressed by the GMIS system.

the CPU's time thereby processing requests faster. For example, if there are ten virtual machines, each one receives one tenth of the real CPU. However, if seven of the ten are in a locked state, then the remaining three receive one-third of the CPU. Thus, these three run (in real time) faster than they did when ten were running.

To try to analyze this circumstance for the uses outlined in this paper, we have assumed that the virtual speeds of VM's are constant and equal. However, when some VM's (including the data-base VM) are allocated a larger share of CPU processing power, they become faster in real time. We assume that each unblocked VM receives the same amount of CPU processing power and at the initial state m machines are running (i.e., the data base machine is stopped if no modeling machines are making requests). 'λ' is the request rate of each modeling VM when there are m VM's running. 'μ' is the service rate at which the data base virtual machine is running when there are m-1 modeling VM and one data base VM running. Thus, we may write the relations:

$$\mu_i = \frac{m}{m-i+1}\,\mu \qquad (i = 1, 2, \ldots, m)$$

$$\lambda_0 = \lambda$$

$$\lambda_i = \frac{m}{m-i+1}\,\lambda \qquad (i = 1, 2, \ldots, m)$$

where i is the number of modeling VM's being blocked. Using a birth/death process model [Drake, 1967], and using a queueing analysis [Little, 1961], we get the following for the response time of the model: where P_i is the steady state probability that there are i modeling machines waiting, and 'N' is the number of modeling machines.

$$T'_{model} = N * \left(\frac{\sum_{i=0}^{m-1} P_i \left(\frac{m-i}{m}\right)}{\sum_{i=0}^{m-1} P_i \left(\frac{m-i}{m}\right) \lambda_i} \right)$$

$$T'_{overhead} = \text{constant}$$

$$T'_{wait\text{-}for\text{-}data} = N * \frac{\sum_{i=1}^{m} iP_i}{\sum_{i=1}^{m} \mu_i P_i}$$

$$T'_{total} = T'_{overhead} + T'_{model} + T'_{wait\text{-}for\text{-}data}$$

Figure 14 illustrates the total time to execute three different models as a function of the number of modeling VM's. Let us consider some of the implications of the above analysis.

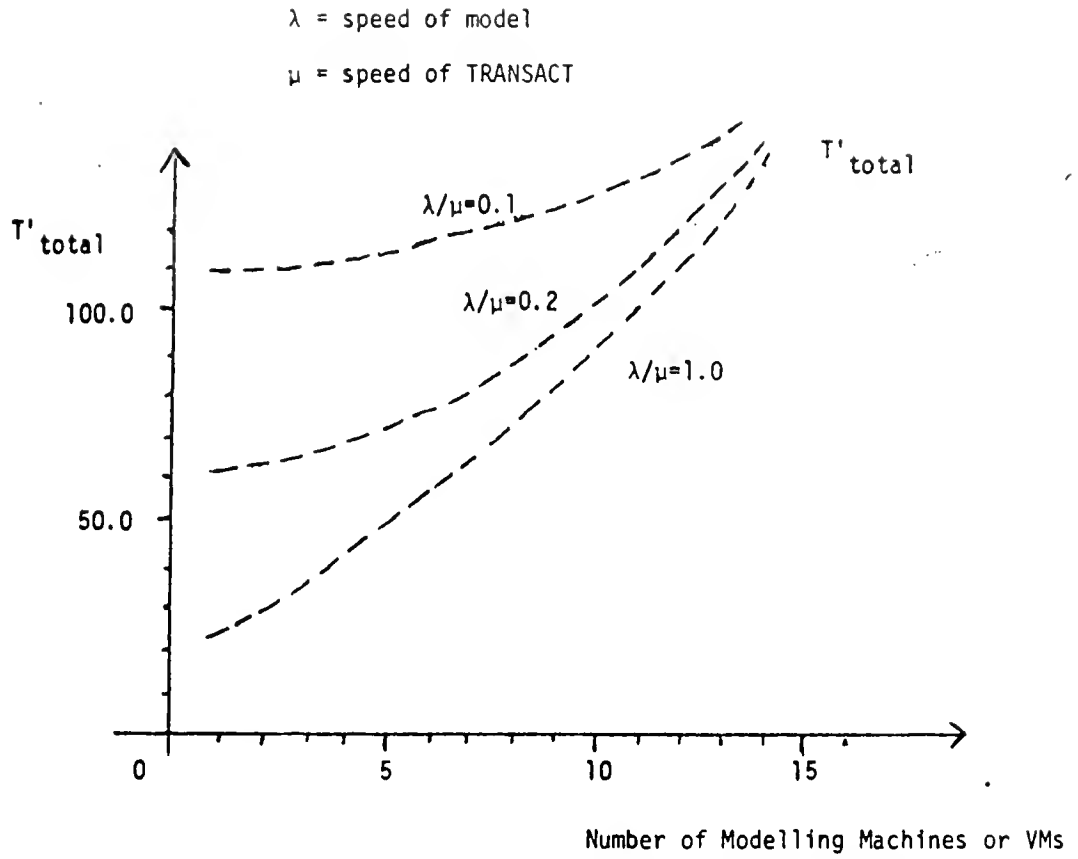

Figure 14. Total Elapsed Times for a VM Configuration

First, for a $\lambda/\mu = .1$, a model executing in a configuration of one modeling machine takes 110 units of time to execute. When the same model, run in an environment of 10 modeling machines all executing similar models, takes approximately 135 units of time to execute -- a degradation of performance of slightly more than 15 percent. Intuitively, λ denotes the speed of the modeling machine, and μ is the speed of the data base machine. Thus a situation where $\lambda/\mu = .1$ indicates that the data base machine is ten times faster than the modeling machine. From the same figure with ratio of $\lambda/\mu = 1$, a model executing with a configuration of one modeling machine takes 20 units of time where with ten machines the same model takes approximately 90 units of time -- over four times longer.

If such a degradation of performance is not tollerable, there are several ways to improve performance. The theoretical study would indicate that increasing μ for a given configuration helps performance. Practically this could be done by changing the processor scheduling algoirthm of VM so that the real processor was assigned to the data base management VM more often, thus speeding it up and increasing μ.

Observing the equation for T'_{total} above, another way of reducing T'_{total} is to reduce $T'_{wait\_for\_data}$. One way to reduce $T'_{wait\_for\_data}$ is to extend the VM architecture of Figure 1 to allow multiple data base machines. In this configuration $T'_{wait\_for\_data}$ could be reduced by locking out all data base machines only when one modeling machine is doing a write. For all read requests the multiple data base machines would operate without locking. Shared locks between machines would have to be created as well as a mechanism for keeping a write request pending until all data base machines can be locked.

A way of improving performance further would be to extend the single locking mechanism used in the above multi data base machine configuration to handle multiple locks. Locks would be associated with groupings of data, e.g., a table. The locking policy would be to have all machines only locked out of a portion of the data when one machine was writing into that portion. Thus requests could be processed simultaneously for reads into tables not being written in and for reads to different tables. Thus adding another real processor to the multiple lock VM configuration could greatly improve performance.

There is a trade off with the multilocking scheme between increases in overhead time in maintaining multiple locks versus increases in wait time for locked data bases. We have not yet extended the theoretical analysis to quantify this trade off.

Other theoretical extensions and analyses of this synchronization model would include extending the model to cover a more common VM operating circumstance -- namely, that where the GMIS system (multiple modeling machines and one data base machine) would have to share the physical machine with other users, also executing under VM, e.g., a payroll program under VS2 under VM, multiple CMS users, etc.

In conclusion, we observe that there may be a degradation in performance with multiple users but that there are mechanisms for ameliorating the effects of this degradation.

4.2 Hierarchical Approach

We have used the design and implementation techniques of hierarchical decomposition extensively in our implementation. The hierarchical approach

has been used in operating systems [Dijkstra, 1968; Madnick and Donovan, 1974] and in file system design [Madnick, 1970]. The essential idea of this approach is to decompose a system into functional levels. Interfaces of each level consist of a series of operators. Each level can only call levels below it.

The levels we are using for the GMIS system are the following:

- a modeling level
- a data definition and data manipulation language level
- a relational level (operators)
- a file system
- the operating system

Further decompositions of the file system level and operating system level are outlined in [Donovan and Jacoby, 1975] and of the relational level in [Madnick, 1975].

The key advantage of this approach is that it reduces complexity by decomposing the problem into a series of manageable sub-problems. As a consequence of this reduction in complexity, the time to implement an entire system is greatly reduced. Another advantage is that the efficiency of the system can be increased. These improvements in efficiency come from the fact that a system so constructed can be analyzed and tuned for performance because each level can be thoroughly understood and analyzed.

For example, as new software algorithms are invented, their place in the hierarchy can be identified and then can be easily incorporated without redesigning the entire system. As new hardware technologies become operational, their relevance to information systems can be assessed within the

the framework of the hierarchy, and incorporated where applicable.

Given inherent parallelism in information systems, the hierarchical approach also can capitalize on new technologies to increase the performance, reliability, and integrity of information systems. An example of such a technological development is the advent of low-cost microprocessors. These devices (which are the "computers" used in hand calculators) are becoming less expensive each year and have the computational capability of many standard computers, e.g., arithmetic and logical operations, memory, and registers. To capitalize on this new technology, each level of the hierarchy could be examined for operators that could be executed asynchronously with each other. These operators, as well as the control logic and synchronization mechanisms, could be performed by multiple microprocessors.

Figure 15 depicts an example of such a hierarchical decomposition using microprocessors where the vertical stacks of boxes denote requests in the form of operators, and each group of horizontal boxes denotes microprocessors to perform the desired operation. At the top of Figure 15 a list of queries enters the system (e.g., the SELECT commands of Figure 7). The microprocessor of level i+2 performs the necessary syntactical analysis and translation to produce a list of relational operators (operators on tables will be discussed in the next section). This list of functions composed of relational operators are processed by the microprocessors at level i+1. They in turn generate a number of requests to read tables stored in the main or secondary memory. Level i receives those requests and generates the appropriate operating system functions to fulfill the request. The last group of microprocessors performs the desired operating system func-

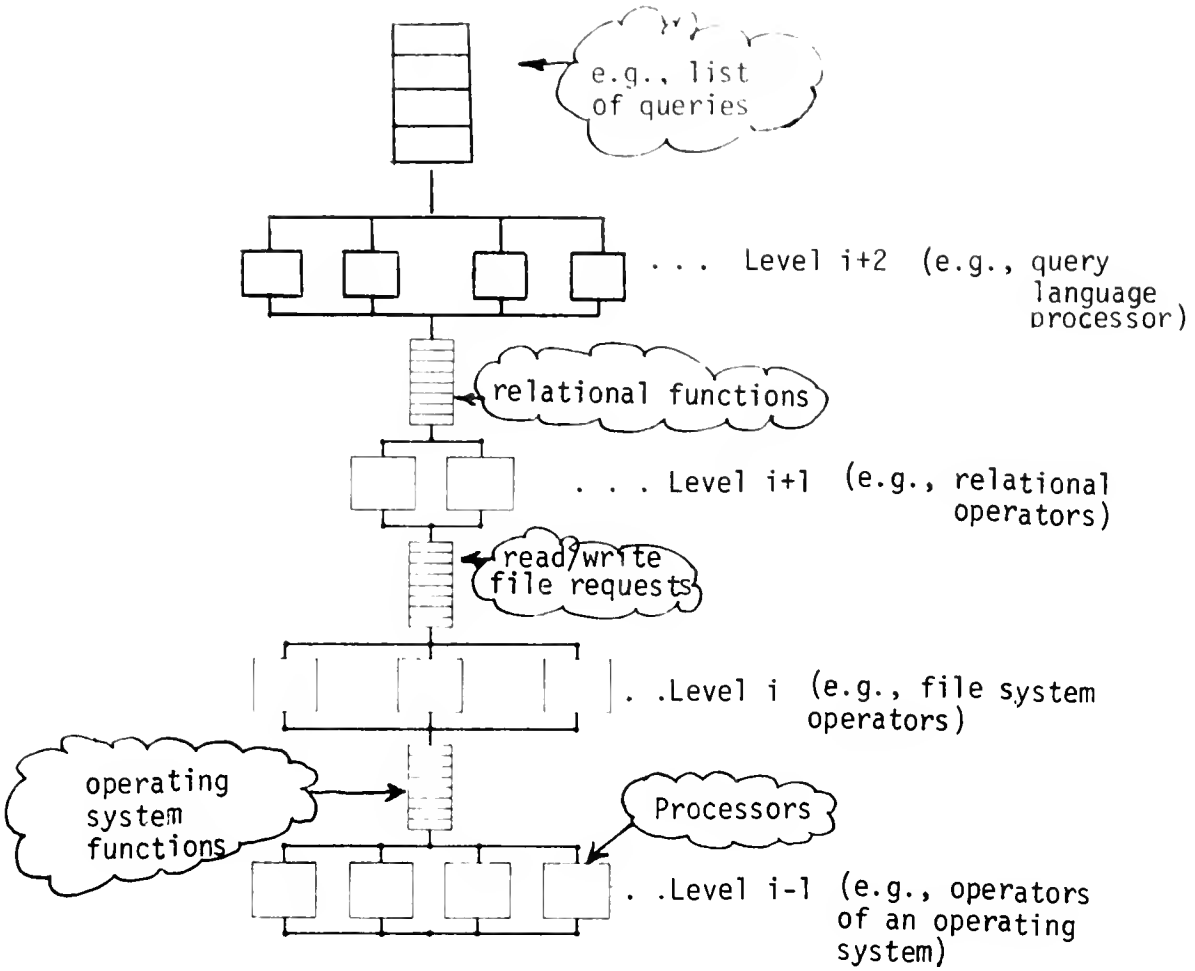

Figure 15. Hierarchical Function Decomposition Using a Microprocessor Complex

tions and passes back the results to level i. The results are used by level i to produce its results, and then passed up to level i+1 until the top level gets all the information to satisfy the query.

One of the properties of implementation using hierarchical function decomposition is that all processors are anonymous and act as interchangeable resources (within a function level). Thus, if a processor malfunctions or must be removed from service, the system can continue to function without interruption. After a reasonable amount of time has elapsed, the higher level processors that had generated requests that were being performed by the defective processor merely need to reissue the same requests. Alternatively, the reissuing of requests could be accomplished automatically by the inter-level request query mechanism.

Although the details are not elaborated in this paper, it can be argued that extensive parallelism, throughput, and reliability can be attained by means of a multiple processor implementation of the hierarhical function decompostion.

4.3 Relational Technology

This section presents an intuitive understanding of relational operators, of the approach, and its usefulness to information systems of the type we address in this paper.

The language that a user would use to query, insert, and update data is called a Data Manipulation Language (DML). The language used to define tables, domains, and characteristics of the data is called a Data Definition Language (DDL). The user of GMIS can view all data stored in the system in the simple form of a table (relation), as in Figure 3. This view of data is called the relational model of data [Codd, 1970].

If one were to view data as being stored in tables, then the process of querying the data could be broken down into two functional levels. The first is composed of mechanisms to recognize the constructs of the query (e.g., a SELECT command), which takes place at level i+2 in Figure 15, and the second where the appropriate operations are performed on the tables to satisfy the SELECT command (level i+1 in Figure 15).

Part of our research has been to determine the "appropriate" operations of level i+1 needed to query, update, and define data. In an early implementation of GMIS we implemented twelve operators [Smith, 1975]. These operators included those of Codd [Codd, 1970] (in some cases modified for use or performance reasons) as well as three additional operators, compaction, difference, and ordering.

4.3.1 Advantages of the Relational Approach

A very attractive aspect of the relational approach is its clear, well-defined interface that fits into the hierarchical approach and hence permits the attainment of all the benefits of the previous section. A distinction should be made (which is not often made in the literature) between the DDL/DML level and the relational operator level. As we shall see, the relational model of data allows us to implement an interactive DDL/DML easily. We recognize that other data models (e.g., network, hierarchical[1], or tree structures) could also be used at a lower level to implement the same DDL/DML, only not in as satisfactory a manner, and with a certain loss of capabilities.

Our experience in using a relational base data management system is that there is a real comparative advantage for its use in systems where the logical data structure keeps changing. Its advantage is the low cost

---

[1] Note the term hierarchical here refers to a tree structure, which is different from the "hierarchical" approach.

of adapting to changing data structures and further, in its use in GMIS, in not having to redo all existing modeling programs. It has a comparative advantage for implementing an interactive DDL/DML. Its comparative advantage, in applications where the types of queries are not all defined before implementation, lies in the inherent property of allowing selective access to any data in the data base. As we will discuss at the end of this section, we recognize the present limitations of the relational approach and do not necessarily advocate it for all data management applications.

4.3.2 Basics of Relational Operators

Let us take an example and demonstrate two relational operators, "restriction" and "projection". Assume that data exists as in Figure 3 and a query is made, "SELECT the model of car that receives 30.2 miles per gallon". The query processor (level i+2 of Figure 15) would translate this query into a series of operators on the table CARSALES. Basically, once the query is recognized there are two operations that could give the desired information: (1) find all entries that have mph equal to 30.2; (2) list the models in those entires.

Figure 16 demonstrates these two operations on the table. All relational operations create new relations. The first operator used is called "restriction", whose function can be intuitively defined as, "produce a relation containing all elements of a table that match particular restricting conditions." Thus, restricting the relation at the top of Figure 16 by the condition MPG = 30.2 produces the relation containing the single tuple:

vega,	1/74,	37600,	30.2

This operaton could be written as follows, where "restriction" is denoted by $\big|_R$.

vega,	1/74,	33600,	30.2

$$= \quad \text{CARSALES} \Big|_R (\text{MPG} = 30.2).$$

The opeation on this newly created relation that we use to produce a list of models is called "projection." Projection's function can be intuitively defined as, "produce a new relation consisting of only those columns specified." Thus "projecting" the relation in the center of Figure 16 on column MODEL produces the single relation at the bottom of Figure 16. The entire relation's function depicted in Figure 16 could be written as follows where projection is denoted by Π,

$$\Pi_{[\text{MODEL}]} \left\{ \text{CARSALES} \Big|_R (\text{MPG} = 30.2) \right\} \qquad (2)$$

Not only can all queries of the DML be translated into a series of relational operators, but all commands of the DDL also can be translated into series of relational opeators on system tables, e.g., the "define table" command merely updates certain system tables. Hence the DDL/DML have their interface to the relational level through a set of powerful operators.

As noted earlier, the result of an operation on a relation is another relation. It is this and other mathematical properties that make this view of data particularly attractive. Another desirable feature is its lack of dependence on the physical structure of the stored data. For example, the above function (2) would apply if there had been many entries where MPG = 30.2; also, the function did not need to be adjusted according to the way these entries were stored.

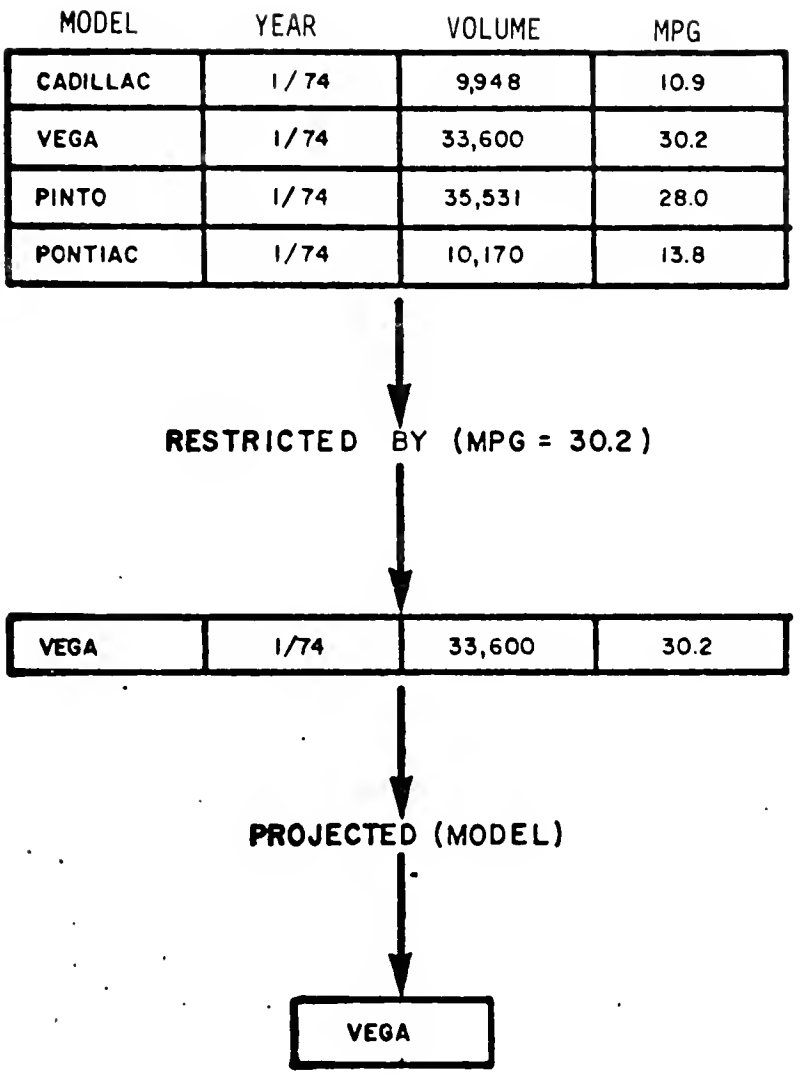

FIGURE 16

RESTRICTION AND PROJECTION OPERATORS

There have been several experimental implementations of the relational view of data. For example, ISG [Smith, 1974], MACAIMS [Goldstein, Strnad, 1971], SEQUEL [Chamberlain, 1974], Colard [Bracchi, 1972], RIL [Fehder, 1972]. In GMIS we are using an extended version of SEQUEL discussed in Section 2.1.

Our experience leads to several conclusions: From a user view the primary advantage of SEQUEL and other relational systems is that they can be interactive, and have a simple, consistent way of viewing data. From an implementor's view the relational implementaion of SEQUEL fits into a hierarchical approach, the operations are consistent, and it provides a framework in which to examine performance. We recognize the present limitations of the experimental SEQUEL for real applications. We list some of those here (not as a criticism of the implementors of SEQUEL, for their purpose was to demonstrate feasibility not an operational system) to guard against the danger that our enthusiasm for this approach will lead to an overoptimistic picture of SEQUEL.[1]

---

[1] Some of the extensions we have had to incorporate in order to make SEQUEL more operational for our applications are the following: (1) Added a facility for multi-user to access the same data base, (2) Added interfaces so that users can use a variety of terminals, (3) Modified SEQUEL to accept the unary + and - operators as prefixes to numeric literals, and to handle DECIMAL constants, (4) Extended SEQUEL implementation restrictions on maximum degree of a table, maximum length of an identifier, and maximum size of a character string constant, (5) Re-wrote output formats for generality, (6) Implemented a macroprocessor capability that allows users to write prepackaged series of queries, (7) Made changes to increase performance, (8) Added the capability to interface modeling and analytic facilities, (9) Enhanced the bulk loading facility, (10) Designed mechanisms for handling null or missing data, (11) Designed backup facilities, (12) Designed security mechanisms, (13) Designed additional SEQUEL operators (e.g., GROUP BY). The documentation of these changes as well as others is found in a NEEMIS Progress Report [M.I.T. Energy Laboratory, 1975].

We feel that an operational relational data management facility needs to be implemented and incorporated into a system that has analytical capabilities. We strongly believe that such a development must be done in close cooperation with real applications. Further, we feel that those applications should be chosen in areas where this technology has a clear advantage, that is, for systems where the problems keep changing (e.g., public policy systems) or where the system is not well-defined (e.g., breadboarding systems), and not to application areas that are currently being satisfactorily met by other approaches.

5. FURTHER RESEARCH

There are several types of research that need to be pursued so that these tools can be made available at reasonable cost, and so they can be used in the most effective manner. Some of that further research has been discussed in the last section.

5.1 Computer and Management Science Research

Optimal Hierarchical Decomposition. To gain insights as to what would be the best hierarchical decomposition, research should be undertaken to define measures that would allow the construction of proofs that a particular decomposition of a hierarchy is optimal.

Performance. Each level of the hierarchy needs both a theoretical study and an empirical study. At each level the impact of new operators should be investigated, along with formalizations for equivalence between sets of these operators and performance implications of new operators. Mechanisms for reducing expressions to equivalent but more efficient expressions should be explored. For example, at the DDL and DML, level algorithms that heuristically take advantage of certain query patterns to make subsequent queries more efficient must be studied. At the relational level ways of simulating certain relational operations when the full operator is not called for must be investigated. Theoretical bounds on computation of relational operations as function of a size of tables must be developed.

Virtual Machines. On the VM interface level there is need for investigation of efficient ways VM's can communicate with each other. On the VM level more knowledgeable processor schedulers need to be developed. And, as was discussed in Section 4.1, work must be done on synchronization and locking policies of multiple VM configurations.

New Technologies. Investigation of the implications of the new technologies (e.g., memory, networks, and microprocessors) on each level in the hierarchy is called for.

Query Languages. On the DML level in addition to the extensions we have made to the SEQUEL language (e.g., multiuser interface, security, additional computational capability, handling larger relations, larger number of entries), new query language constructs ought to be investigated. Realistic and operational implementations of a relational query language should be undertaken.

Syncronization and Interlocks. Various interlock mechanisms must be used in an information system to coordinate various independent update operations. It is necessary to develop interlock techniques and policies that lend themselves to a highly decentralized implementation without adversely affecting performance or reliability. For example, under what condition and for how long are the modeling machines locked out of the data base machine? Is the data base machine just a catalogue for data stored in the decentralized data base machines? If so, what are the performance implications of always accessing data stored in a remote machine? Or is the accessed data brought up to the data base or modeling machine in which case what are the updating policies? What sort of hardware can best support the proposed hierarchical structure and system structure?

5.2 Studies of the Economics of Information System Design

Traditional measures of performance (e.g., throughput, system utilization, response time, turnaround time, etc.) are potentially misleading and may be irrelevant for the class of information systems addressed here.

These measures address themselves only to the variable costs of an information system. In the development of an information system there are fixed costs (analysis cost, design cost, implementation cost of the software, as well as the hardware costs) as well as variable costs (costs of queries, execution of models and analytical functions). Much more research is needed on the overall costs of information systems, on more general concepts of "performance," and on the types of studies that should be done in choosing a software system appropriate to the particular task at hand.

To illustrate the point, take the simple example of the design of an inventory control system for a large manufacturer on the one hand, and a system of roughly the same character and complexity to serve as federal energy policy on the other. The costs of developing such systems using different sets of software tools are illustrated in Figure 17. The solid lines show the fixed and variable costs of constructing either of these two systems using a conventional package, say IMS. The dashed line shows the cost of the same systems with tools such as those provided by GMIS. For the more flexible GMIS-type system the fixed costs (and thus the time to build the system) are much lower, but this advantage comes at the expense of increased variable costs.[1]

Provided the purposes for the two systems are well known and the operating assumptions are fixed, the two systems break even at Point A.

---

[1] It is likely that hardware will eventually be developed to support this sort of system, and variable costs will be substantially reduced [Madnick, 1975].

_____ System constructed with conventional information management tools.

- - - - - - - - System constructed with GMIS-type tools.

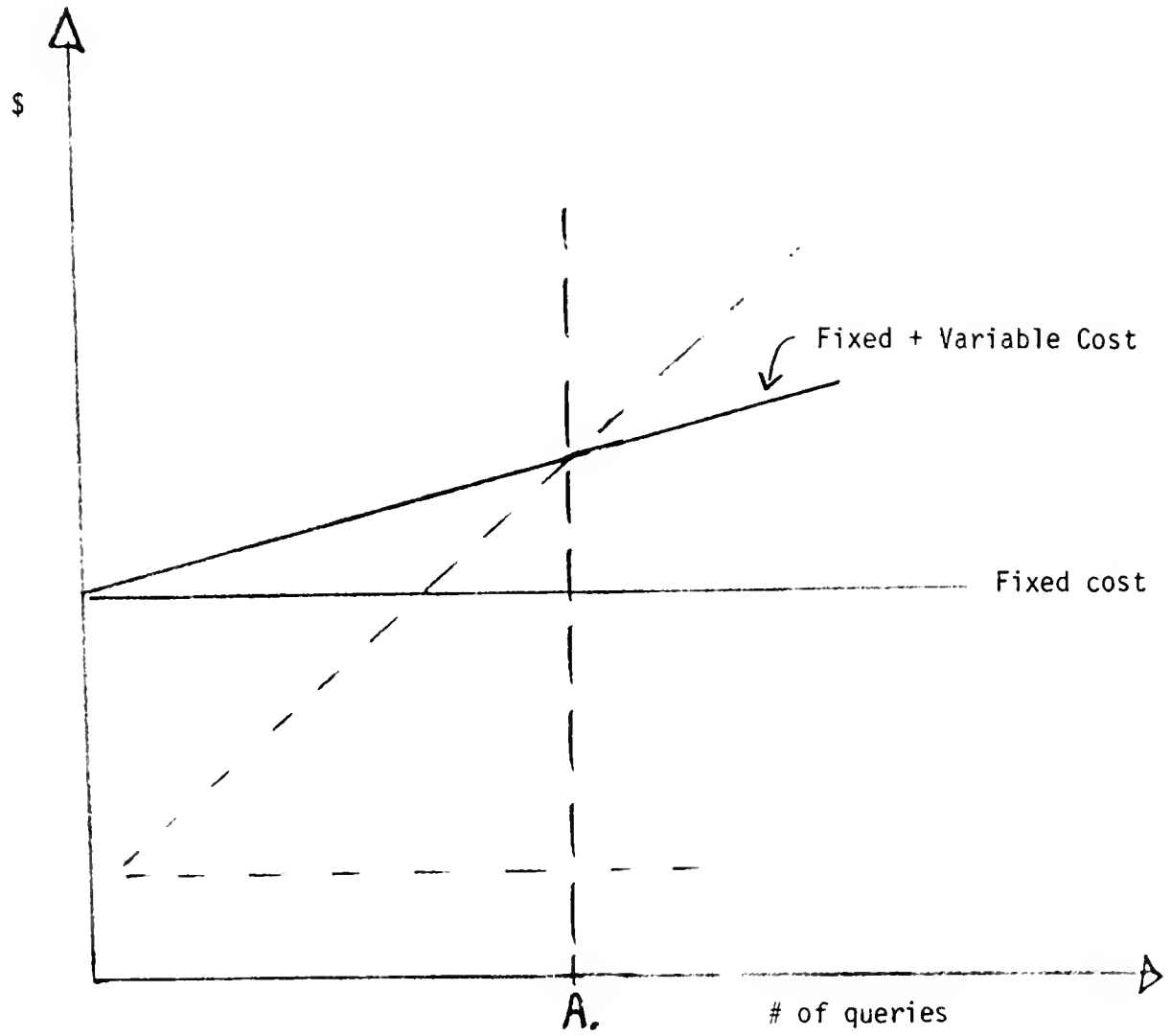

Figure 17

Fixed costs versus variable costs.

If the application anticipates a large volume of queries as the inventory example might, then the conventional approach is preferred.[1]

Of course, to the extent that information system purposes and operating conditions change over time, the fixed costs of each system are multiplied by some factor -- a condition which greatly favors the types of tools discussed above.

The economics of these choices are poorly understood, and the development of better indices of system "performance" is a high priority item in information systems research. When these comprehensive indices of performance are developed, however, we expect that systems like GMIS will receive high marks for a wide variety of applications. Already the system is proving its worth in application to New England energy problems, and to several areas of policy research in the M.I.T. Energy Laboratory. We hope for continued progress on the issues and problems that remain, and look forward to a new generation of information management and analysis systems that are better suited to the fast-moving pace of many corporate and public problems.

---

[1] A GMIS-type system may still be a useful tool (as a breadboarding system) in the optimization of the design of the data management facility, even with the implementation to be carried out with some other package.

REFERENCES

Association for Computing Machinery, "DBTG CODASYL," New York, 1971.

Bracchi, G. et. al.: "A Language for a Relational Data Base Management System," Proceedings of 5th Princeton Conference on Information Science, 1972.

Buzen, J. P., P. Chen, and R. P. Goldberg: "Virtual Machine Techniques for Improving System Reliability," Proceedings of the ACM Workshop on Virtual Computer Systems, March 26-27, 1973.

Chamberlain, D. D. and R. F. Boyce: "SEQUEL: A Structured English Query Language," Proceedings of 1974 ACM/SIGFIDET Workshop, 1974.

Cincom Systems, Inc.: TOTAL Reference Manual, Edition 2, Version II, Cincinnatti, Ohio, 1974.

Codd, E. F.: "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, vol. 13, no. 6, June 1970, pp. 377-387.

Dijkstra, E.: "T.H.E. Multiprogramming System," Communications of the ACM, May 1968.

Donovan, J. J.: Systems Programming, McGraw-Hill, New York, 1972.

Donovan, J. J.: "Use of Virtual Machines in Infomration Systems," Report CISR-10, M.I.T. Sloan School of Management Working Paper No. 790-75, May 1975.

Donovan, J. J. and H. D. Jacoby: "A Hierarchical Approach to Information System Design," Report CISR-5, M.I.T. Sloan School of Management Working Paper No. 762-75, January 1975.

Donovan, J. J. and S. E. Madnick: "Application and Analysis of the Virtual Machine Approach to Computer System Security and Reliability," IBM Systems Journal, May 1975.

Drake, A. W.: Fundamentals of Applied Probability Theory, McGraw-Hill, New York, 1967.

Dynamics Association: XSIM User's Guide, Cambridge, Mass., 1974.

Environmental Protection Agency, 1975 Gas Mileage Guide for New Car Buyers, 2nd Edition, Washington, D. C., January 1975.

Environmental Protection Agency, 1974 Gas Mileage Guide for New Car Buyers, Washington, D. C., January 1974.

Fehder, P. C.: "The Representation of Indepndent Language," IBM Technical Report RJ1121, November 1972.

Goldberg, R. P.: Architecture of Virtual Machines," Proceedings 1973 AFIPS National Computer Conference, vol. 42, pp. 309-318, 1973.

Goldberg, R. P.: "Survey of Virtual Machine Research," Computer, vol. 7, no. 6, June 1974, pp. 34-35.

Goldstein, I. and A. Strnad: "The MACAIMS Data Management System," M.I.T. Project MAC TM-24, April 1971.

Gutentag, L. M.: "GMIS: Generalized Management Information System -- an Implementation Description," M.S. Thesis, M.I.T. Sloan School of Management, June 1975.

Hall R.: "TSP Manual," Harvard Technical Report No. 12, Harvard Institute of Economic Research, Cambridge, Mass., April 1975.

Hatfield, D. J.: "Experiments on Page Size Program Access Patterns, and Virtual Memory Performance," IBM Journal of Research and Development, vol. 16, no. 1, pp. 58-66, January 1972.

IBM: "IBM Virtual Machine Facility/370: Introduction," Form Number GC20-1800, White Plains, New York, July 1972.

IBM: "IBM Command Language Guide for General Users," order no. GC20-1804-2, White Plains, New York, 1974.

IBM: "IMS,"Form Number H20-0524-1," White Plains, New York, 1968.

IBM: "APL Econometric Planning Language," Form Number SH20-1620, Armonk, New York, (Product # 5796PDW), 1975.

Iverson, K. E.: A Programming Language, John Wiley & Sons, 1962.

Little, J. D. C.: "A Proof of the Queueing Formula: $L = \lambda\omega$," Operations Research 9, 1961, pp. 383-387.

Madnick, S. E.: "Design Strategies for File Systems," M.I.T. Project MAC TR-78, October 1970.

Madnick, S. E.: "INFOPLEX -- Hierarchical Decomposition of a Large Information Management System Using a Microprocessor Complex," Proceedings of 1975 AFIPS National Computer Conference, 1975.

Madnick, S. E.: "Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach," Modern Data 2, 3, March 1969, pp. 34-36.

Madnick, S. E. and J. J. Donovan: Operating Systems, McGraw-Hill, New York, 1974.

M.I.T. Energy Laboratory "Energy Indicators," Final Working Paper submitted to the F.E.A. in connection with a study of Information Systems to Provide Leading Indicators of Energy Sufficiency, Working Paper No. MIT-EL-75-004WP, June 1975.

M.I.T. Energy Laboratory, "GMIS Primer," Working Paper No. MIT-EL-75-012, September, 1975.

M.I.T. Energy Laboratory, "Progress Report on NEEMIS Task Order No. 4," Working Paper No. MIT-EL-75- , September 1975.

Morrison, J. E.: "User Program Performance in Virtual Storage Systems," IBM Systems Journal, vol. 12, no. 3, 1973, pp. 34-36.

MRI Systems: "System 2000 Reference Manual," Austin, Texas, 1974.

National Bureau of Economic Research: TROLL Reference Manual, Technology Square, Cambridge, Mass., 1974.

Pakin, S.: "APL/360 Reference Manual," Science Research Associates, Chicago, 1972.

Parmelee, R. P., T. I. Peterson, C. C. Sullivan, and D. S. Hatfield: "Virtual Storage and Virtual Machine Concepts," IBM Systems Journal, vol. 11, no. 2, 1972, pp. 99-130.

Popek, G. J. and C. Kline: "Verifiable Secure Operating Systems Software," Proceedings of 1975 AFIPS National Computer Conference, 1975.

Schober, F.: "EPLAN -- An APL-Based Language for Econometric Modeling and Forecasting," IBM Philadelphia Scientific Center, 1974.

Smith, G. M.: "Internal Intermediate Langauge, Version 2," M.I.T. Sloan School of Management, Management Science Group, November 1974.

Satty, T. C.: Elements of Queueing Theory, McGraw-Hill, New York, 1961.

WARD's Communications, Inc.: WARD's 1975 Automotive Yearbook, 37th Edition, Detroit, Michigan, 1975.